Hochschule Konstanz
Fakultät Informatik

Machine Learning:: Trees and Ensemble Methods

Dozenten: Prof. Dr. M. O. Franz, Prof. Dr. O. Dürr

Remark: Much of the material have been developed together with Beate Sick (ZHAW/Uni Zurich).

Content

- Decision Trees
  - Decision Trees are the basis for recent developments
- Bagging and Random Forest
    - Random Forest
- Boosting
  - Ada-Boost (Algorithm)
  - Gradient Boosting

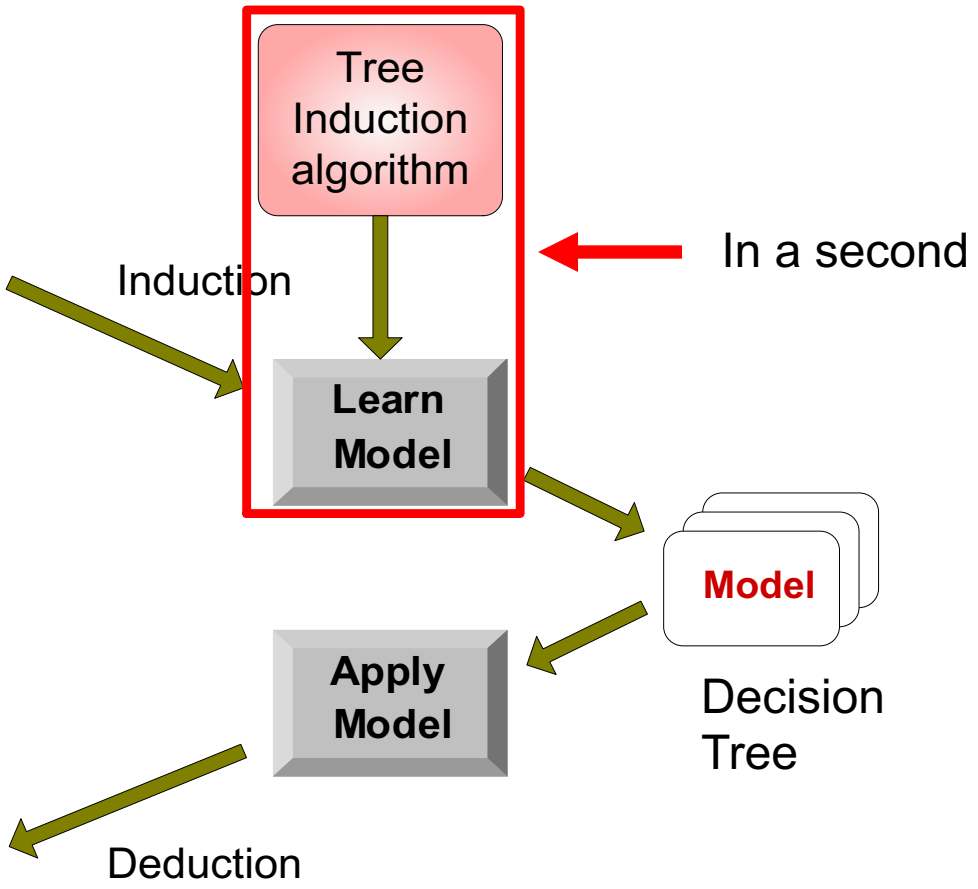
Decision Tree Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

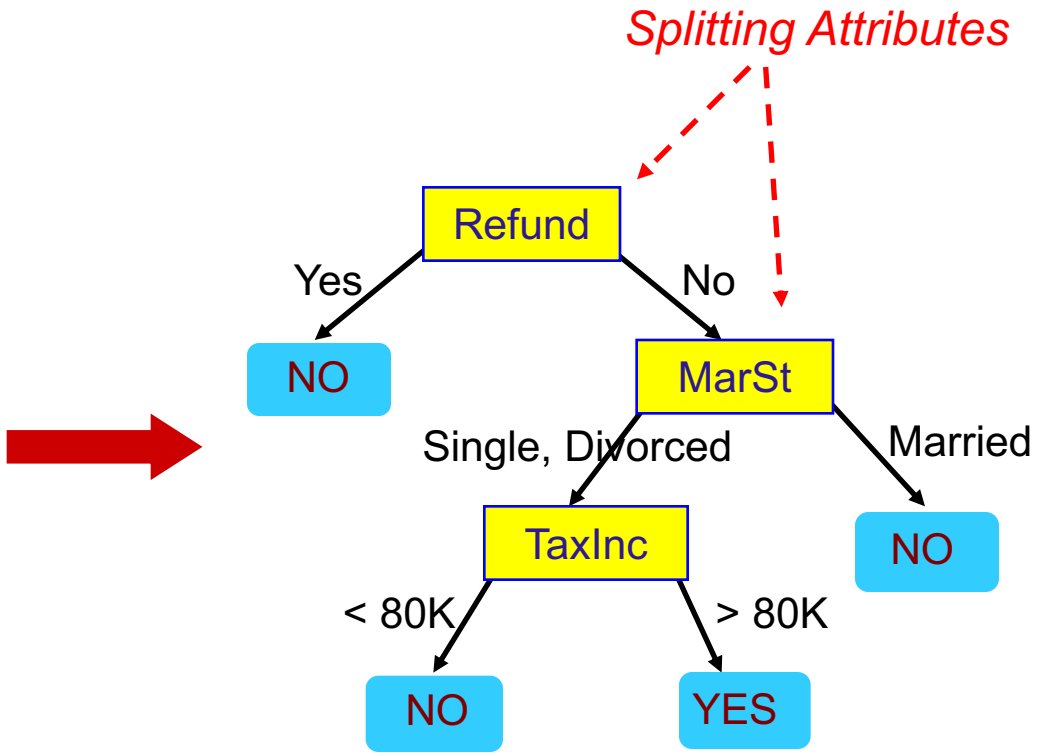
Test Set



Example of a Decision Tree for classification

Tid	Refund (categorical)	Marital Status (categorical)	Taxable Income (continuous)	Cheat (class)
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

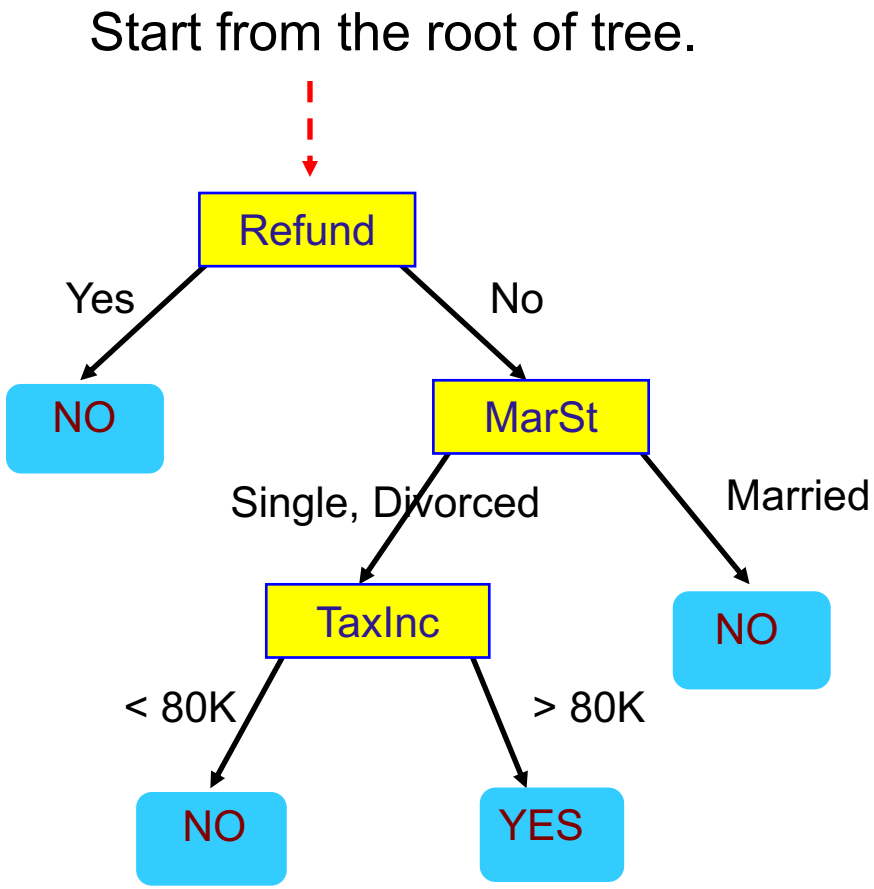
Training Data



Model: Decision Tree

Source: Tan, Steinbach, Kumar

Apply Model to Test Data

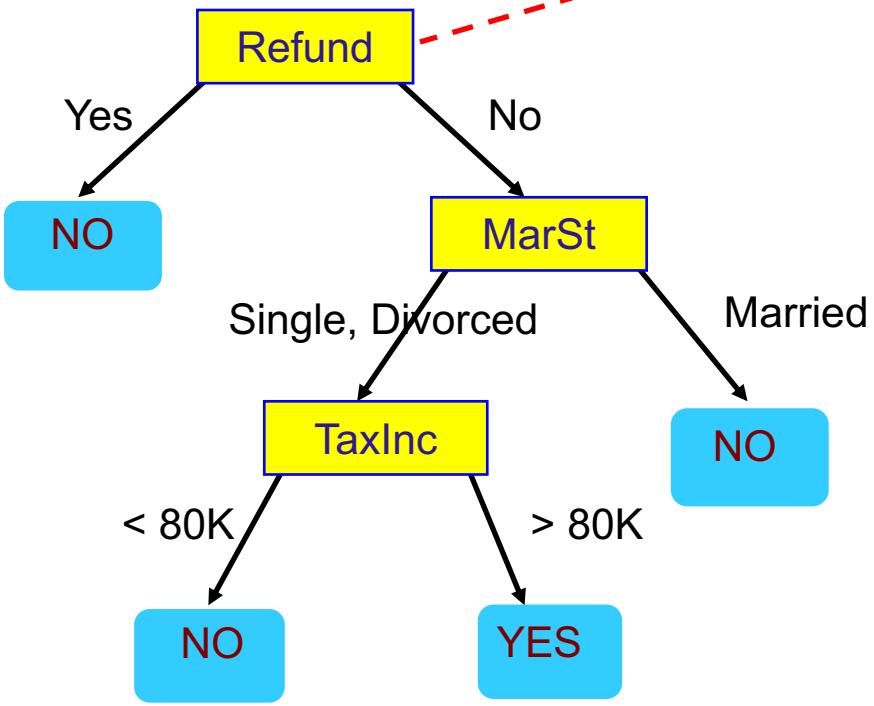


Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Source: Tan, Steinbach, Kumar

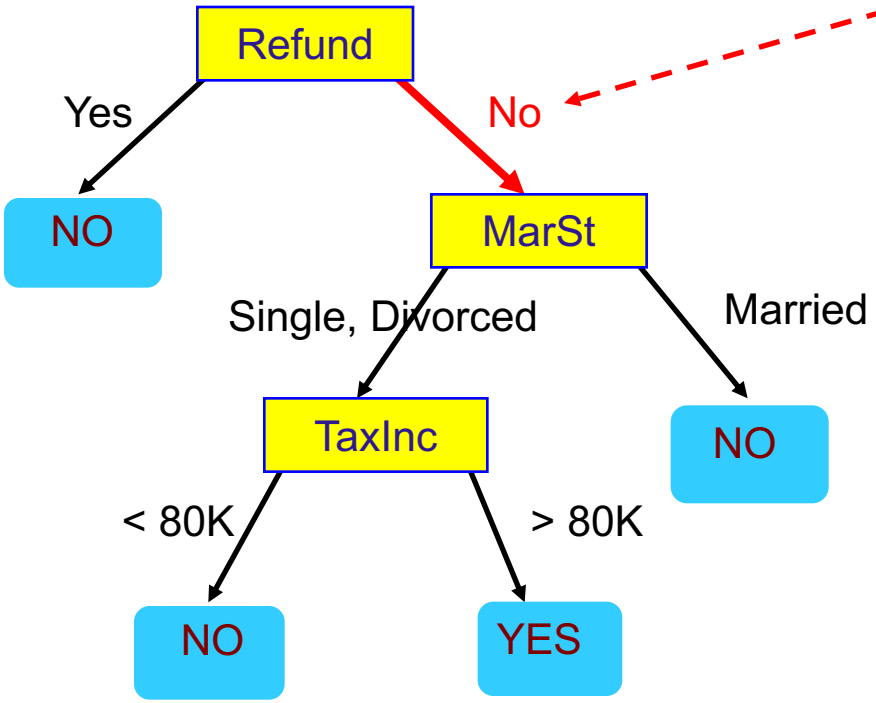
Apply Model to Test Data



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

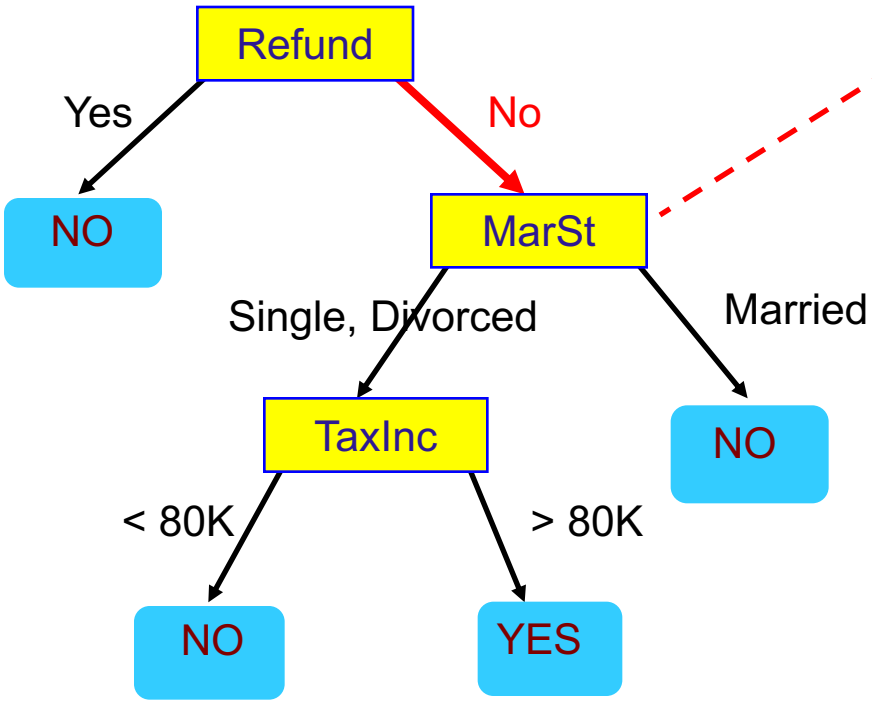
Apply Model to Test Data



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data



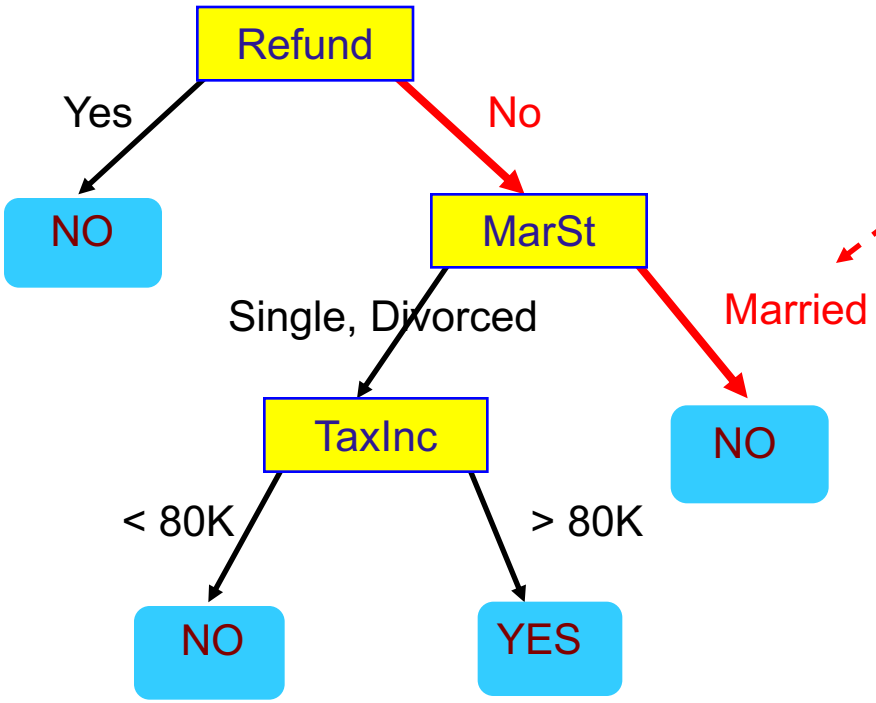
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data

Test Data

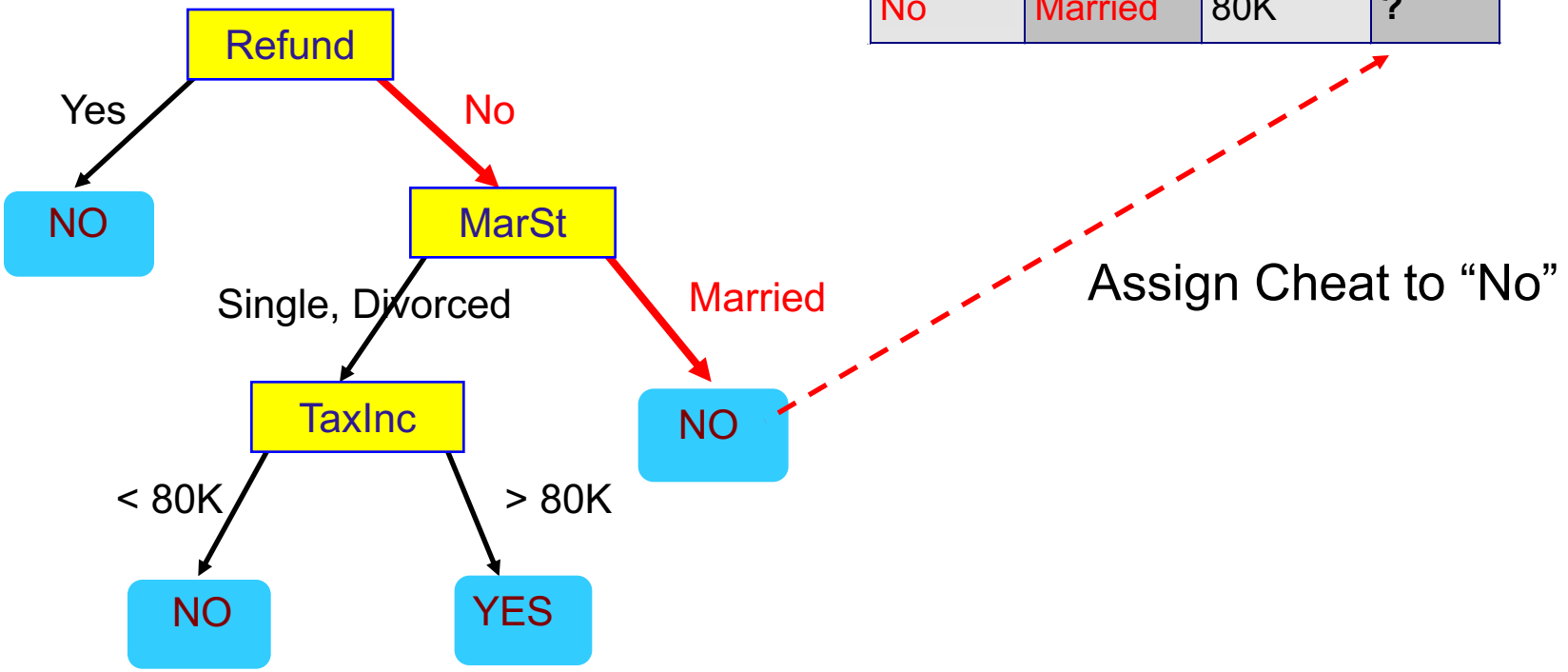
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply Model to Test Data

Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

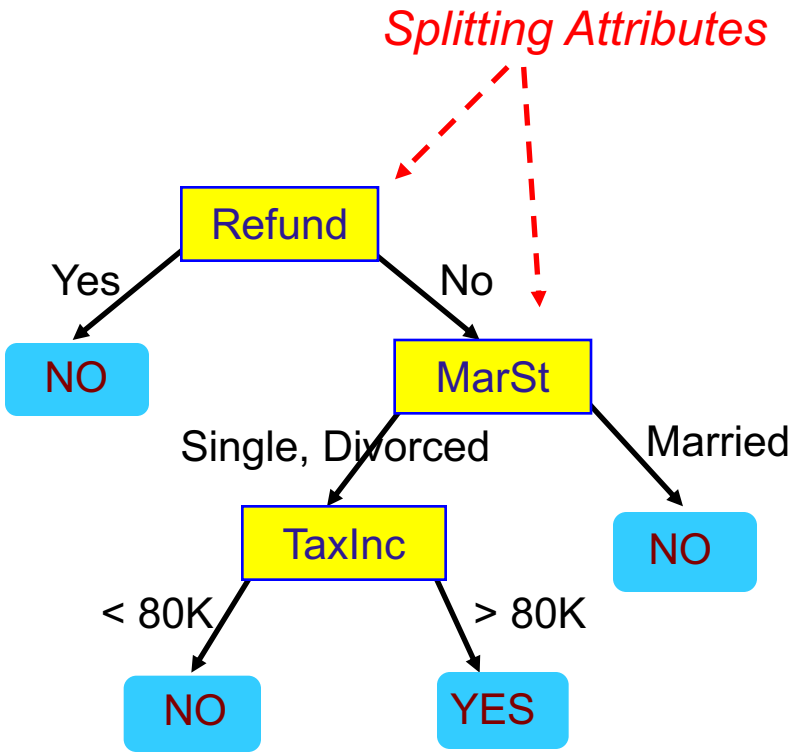


Source: Tan, Steinbach, Kumar

Ambiguity of learned tree:
Example of a Decision Tree for classification

	categorical	categorical	continuous	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

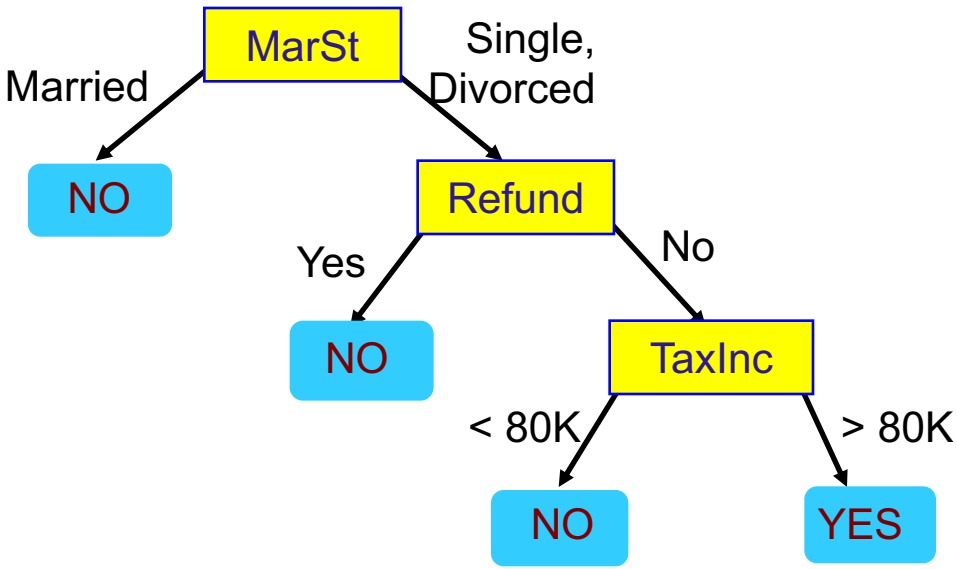
Training Data



Model: Decision Tree

Ambiguity of learned tree: Example of a Decision Tree for classification

	categorical	categorical	continuous	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

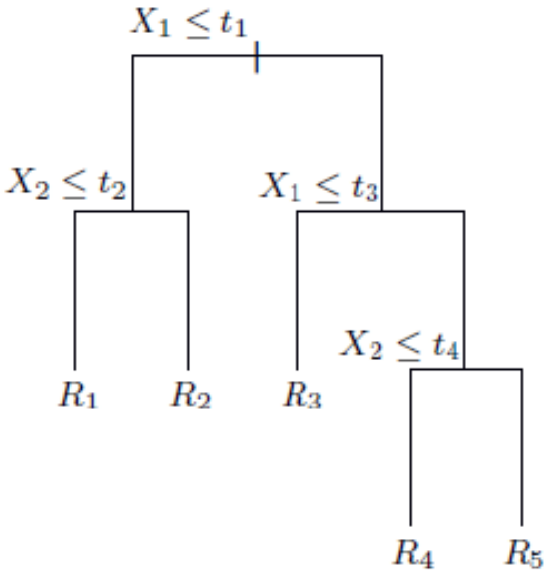


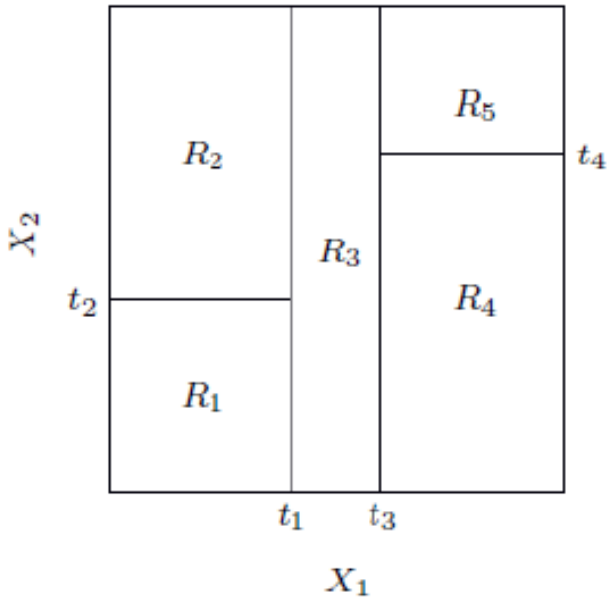
There could be more than one tree that fits the same data!

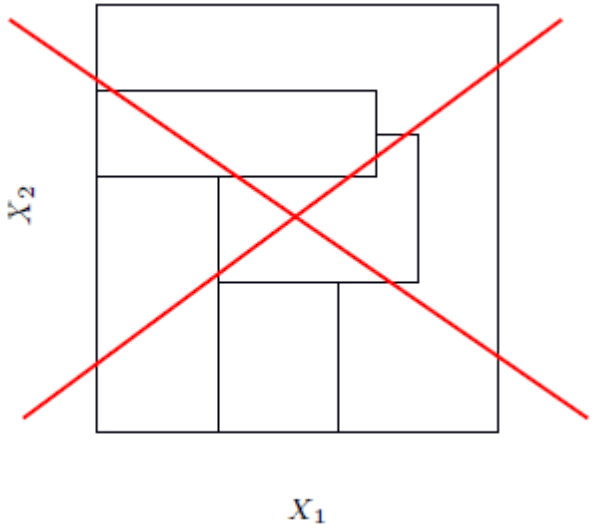
Source: Tan, Steinbach, Kumar

Trees are also called recursive partitioning

- There many approaches
  - C 4.5 / C5 Algorithms, SPRINT
- Here top down splitting until no further split possible (or other criteria)







Also only straight lines

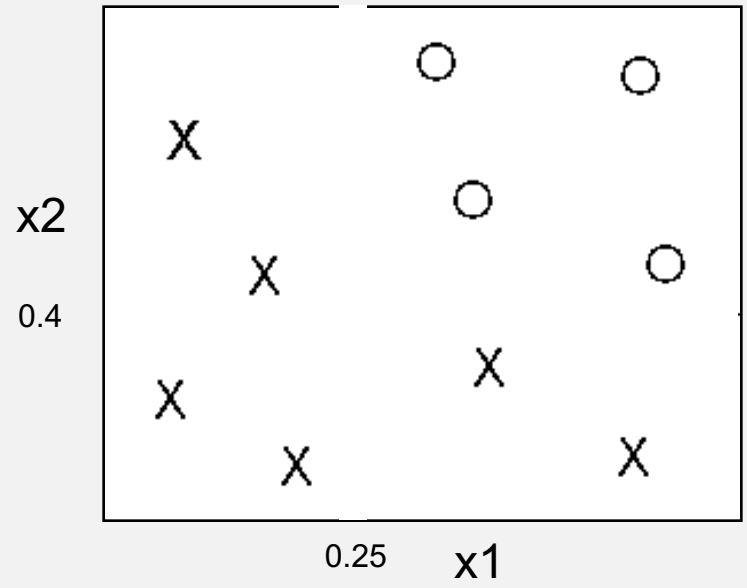
How to train a classification tree: your turn

- Starting with a single region -- i.e., all given data
- At the m-th iteration:

for each region R
  for each attribute x_j in R
    for each possible split s_j of x_j
      record change in score when we partition R into R^l and R^r
Choose (x_j , s_j) giving maximum improvement to fit
Replace R with R^l; add R^r

Score - Node impurity (in a sec)

- Draw 2 splits to separate the data.
- Draw the corresponding tree

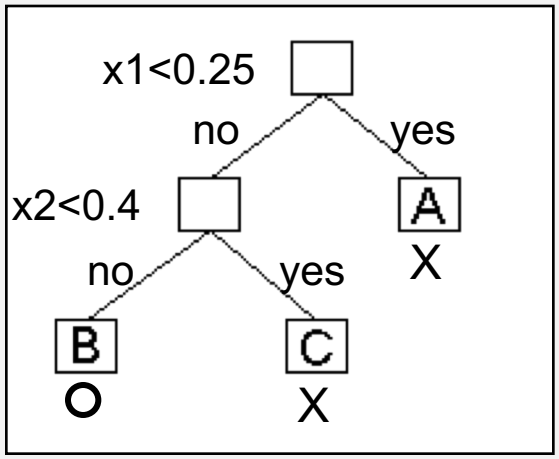


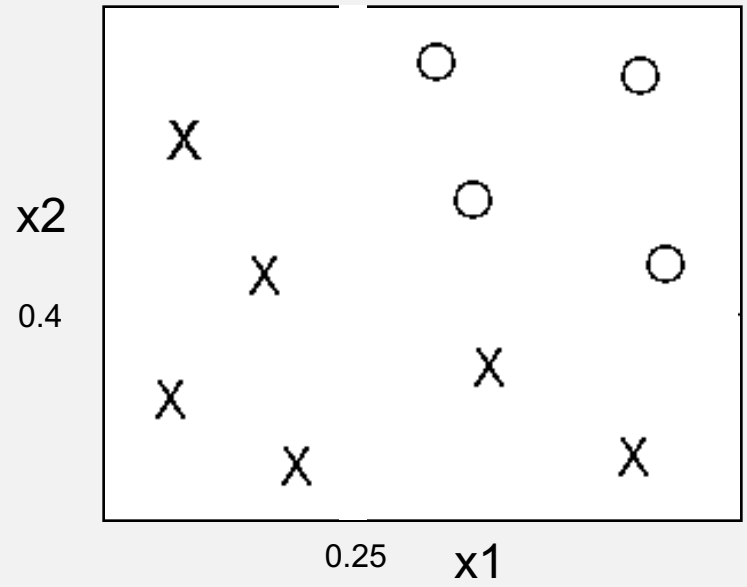
How to train a classification tree: [solution]

- Starting with a single region -- i.e., all given data
- At the m-th iteration:

for each region R
  for each attribute x_j in R
    for each possible split s_j of x_j
      record change in score when we partition R into R^l and R^r
Choose (x_j , s_j) giving maximum improvement to fit
Replace R with R^l; add R^r

Score - Node impurity (in a sec)



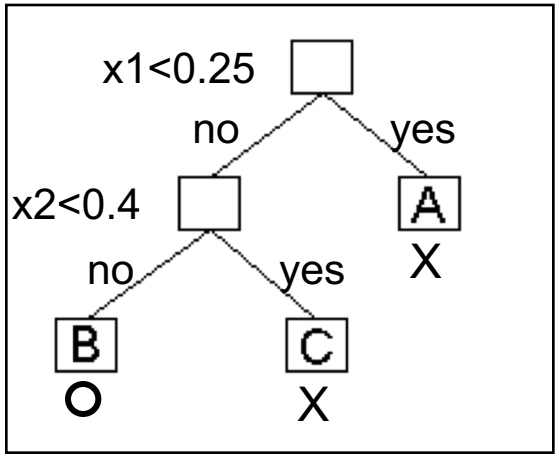


Putting it all together

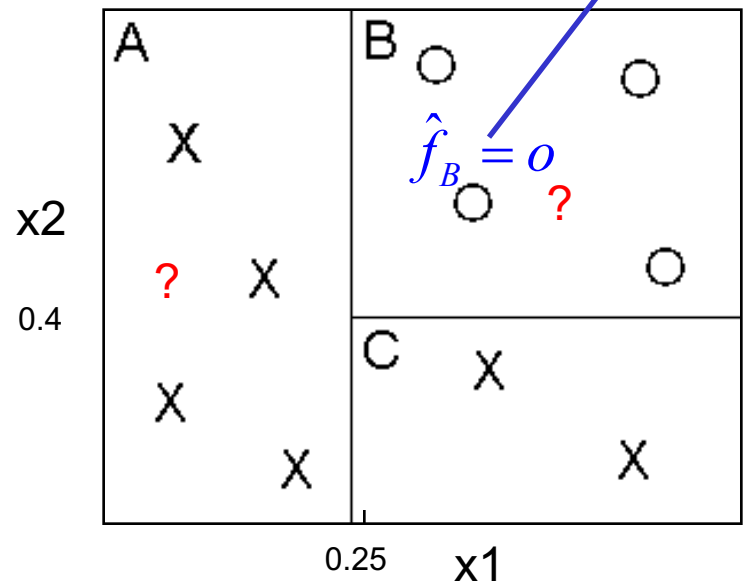
- Starting with a single region -- i.e., all given data
- At the m-th iteration:

for each region R
  for each attribute x_j in R
    for each possible split s_j of x_j
      record change in score when we partition R into R^l and R^r
Choose (x_j, s_j) giving maximum improvement to fit
Replace R with R^l; add R^r

Score - Node impurity (in a sec)

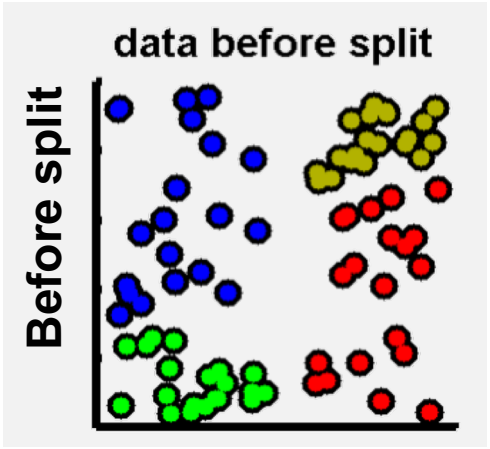


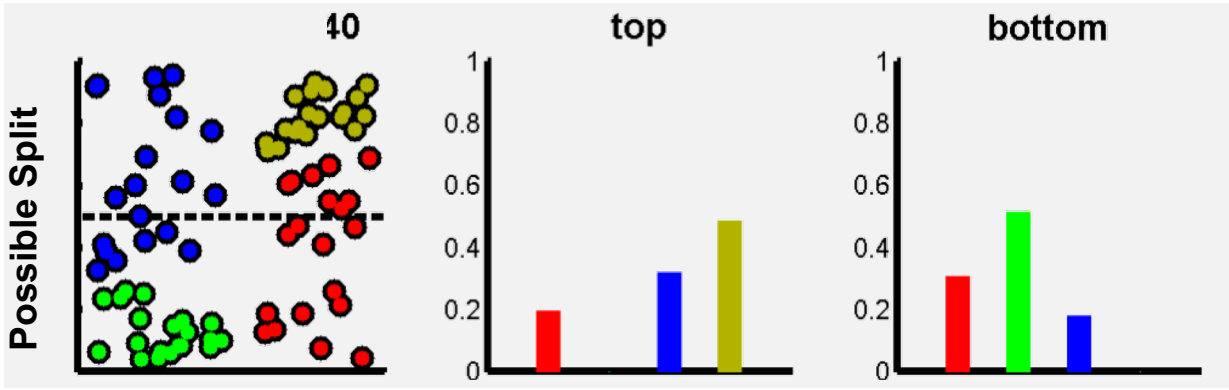
Model class label is given by the majority of all observation labels in the region

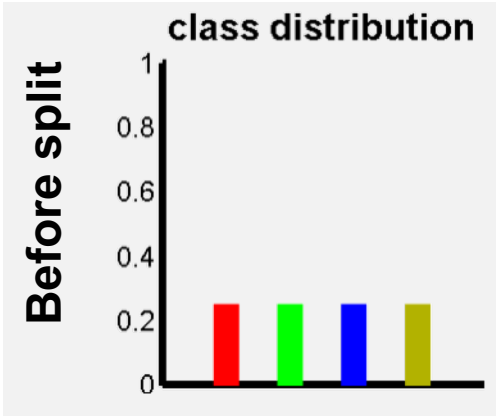


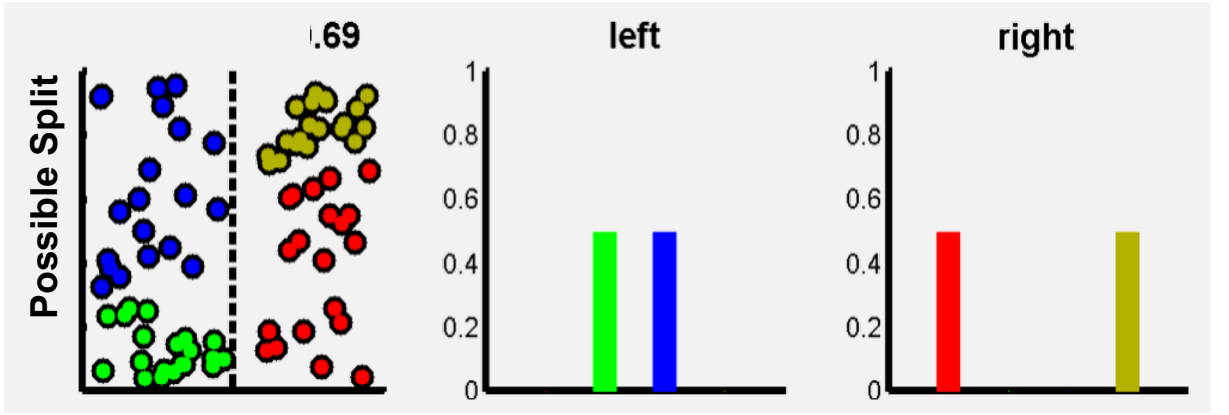
Construction of a classification tree: Minimize the impurity in each node

We have n_c=4 different classes: red, green, blue, olive









The most common impurity measures

p_i is the relative frequency of class $i = 1 \ldots C$

Gini Impurity* (default rpart)

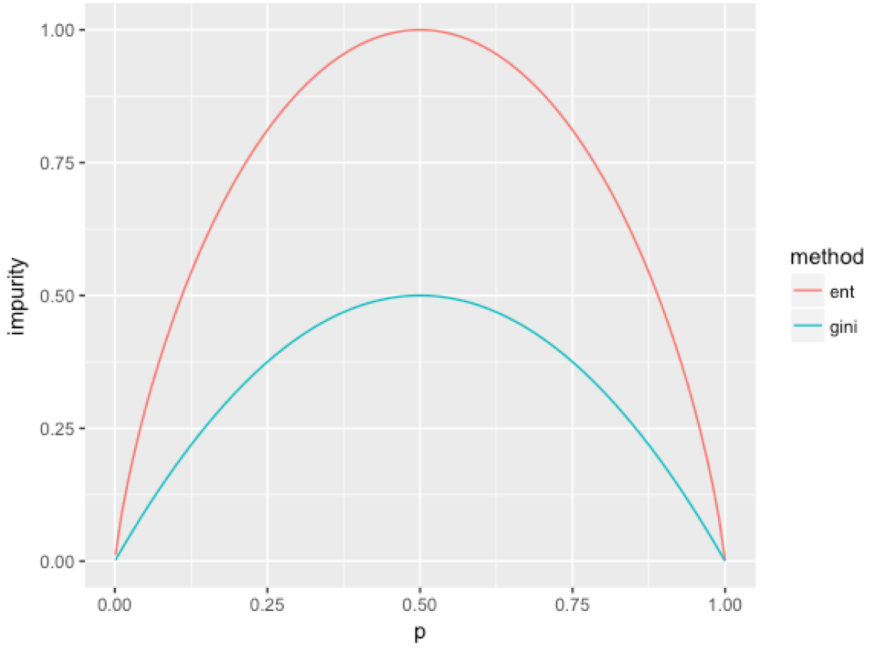
$$\text{Gini} = 1 - \sum_i^C p_i^2$$

Entropy:

$$\text{Entropy} = -\sum_i^C p_i \log_2(p_i)$$

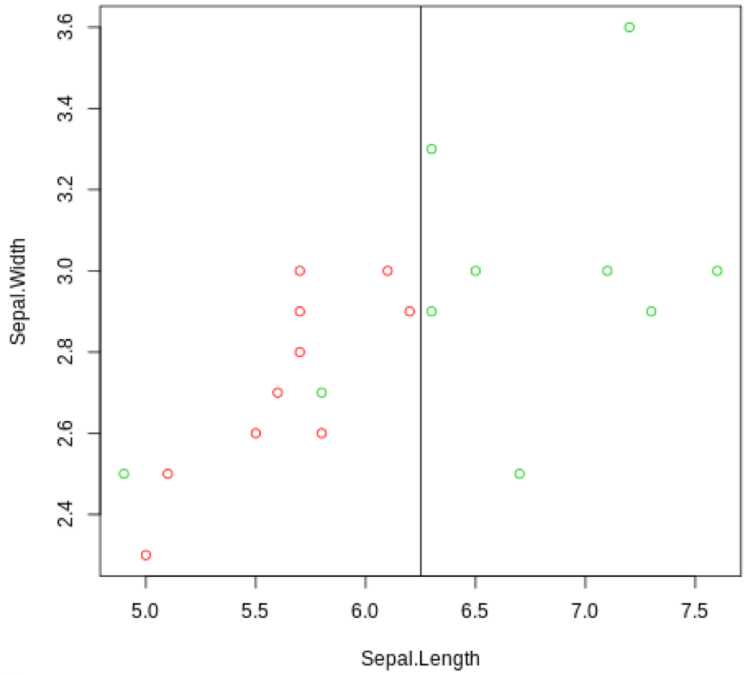
For C=2 classes
p probability for the one class
$1 - p$ for the other

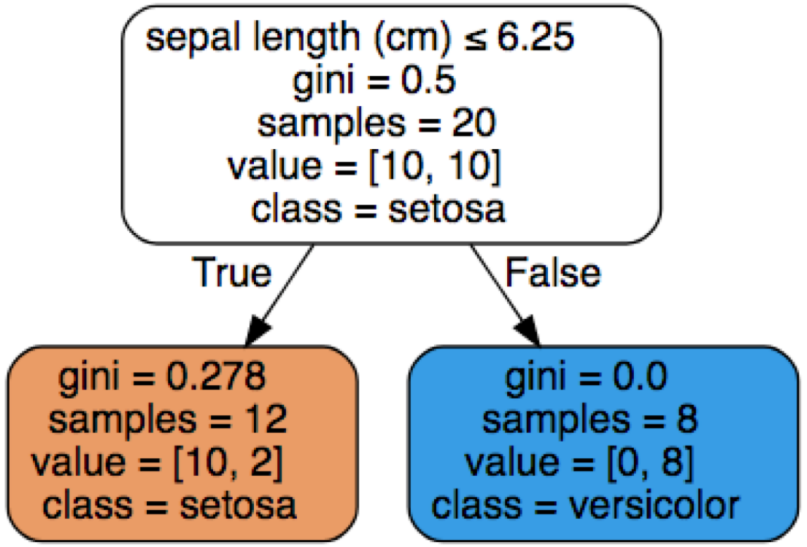
$$\text{Gini} = 1 - p^2 + (1 - p)^2$$



*Not to be confused with Gini coefficient

Example: One split by hand






```
giniroot = 0.5 # p = 0.5
giniright = 0  # p = 0.0
p = 2.0/12
ginileft = 1 - p**2 - (1-p)**2
ginileft
```

Out[19]:

```
0.2777777777777777
```

$$\frac{N_t}{N} \cdot (\text{giniroot} - \frac{N_{tr}}{N_t} \cdot \text{giniright} - \frac{N_{tL}}{N_t} \cdot \text{ginileft})$$

```
impurity_decrease =  1 * (giniroot - 12/20 * ginileft - 8/20.0 * giniright)
impurity_decrease
```

```
0.33333333333333337
```

https://github.com/oduerr/ml-playground/blob/master/python/d_tree/Decision_tree.ipynb

Decision Trees for Regression

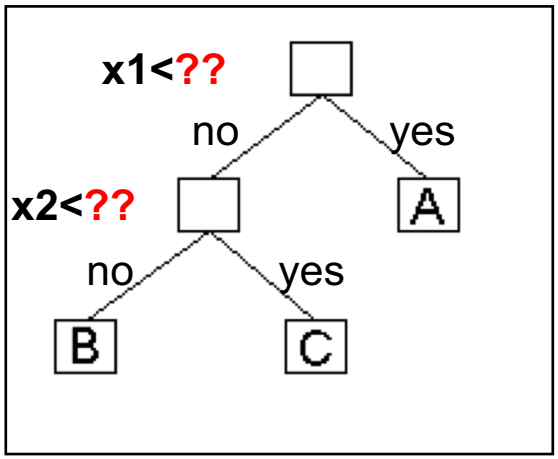
How to find the tree structure of a regression tree?

- Starting with a single region -- i.e., all given data
- At the m-th iteration:

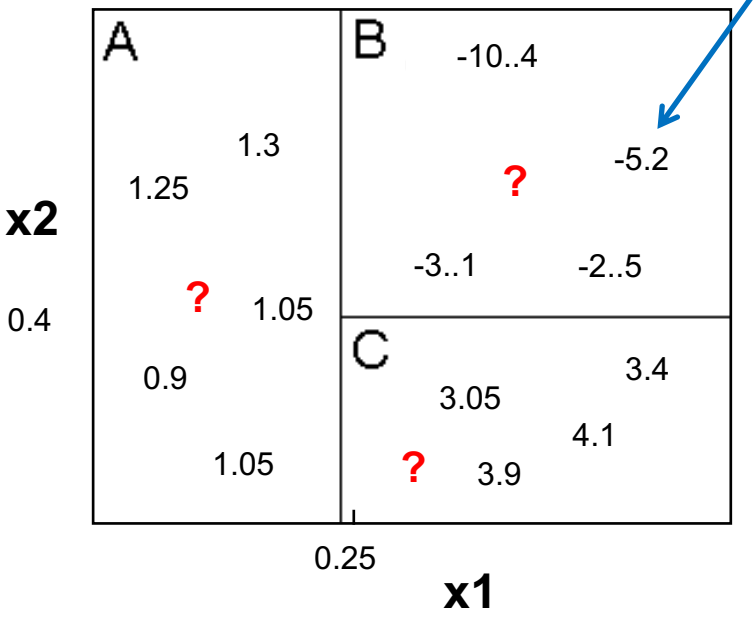
for each region R
  for each attribute x_j in R
    for each possible split s_j of x_j
      record change in score when we partition R into R^l and R^r
Choose (x_j, s_j) giving maximum improvement to fit
Replace R with R^l; add R^r

Score:

- **E.g. based on MSE (mean squared error)**
- **E.g. based on**

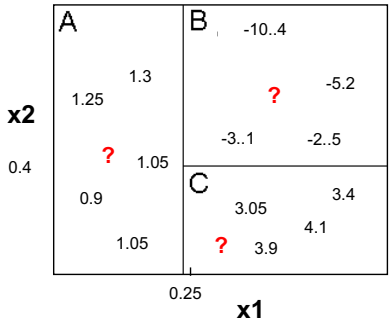


numbers indicate values of **continuous outcome y**



What to report in the regions?

Best value in region



- Say we have the regions
  - What single number to report in a certain region?
- If loss is MSE and we have values y_i then mean reduces the loss
- Proof (optimal value *c* for MSE is mean)

$$loss(c) = \sum_{i=1}^{N} (y_i - c)^2$$

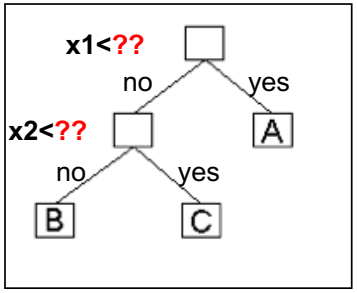
$$\frac{\partial loss(c)}{\partial c} = 0$$

$$\sum_{i=1}^{N} (-2) \cdot (y_i - c) = 0$$

$$N \cdot c = \sum_{i=1}^{N} y_i \rightarrow c = \frac{1}{N} \sum_{i=1}^{N} y_i = mean(y_i)$$

Given a region, we now best single value for that region (it's the mean when MSE)

Best Boundary for MSE



for each attribute x_j in R

  for each possible split s_j of x_j

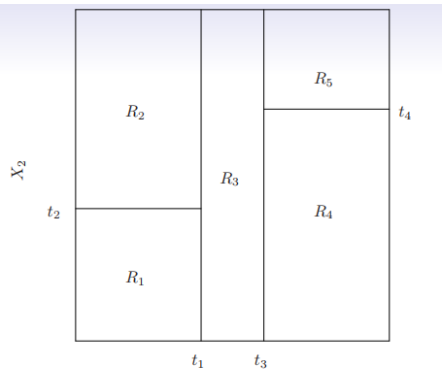
$$\min_{j,\,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right]$$

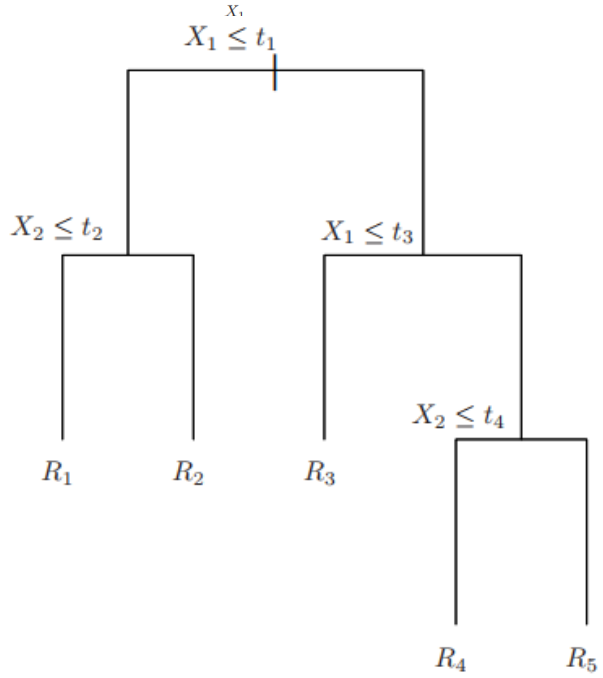
c_1 and c_2 are the mean values (see last slide)

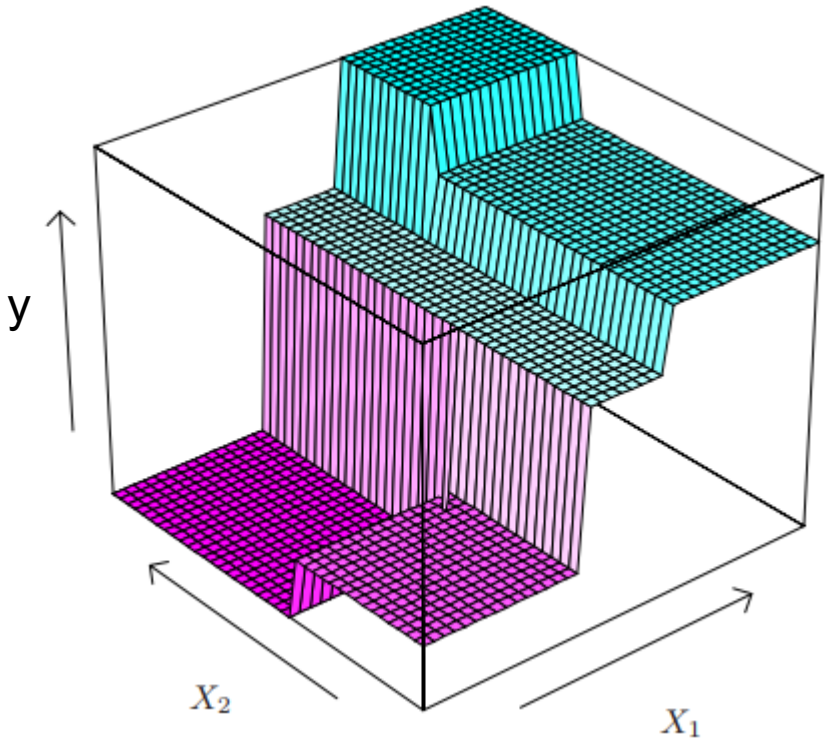
Other criteria:

- MSE is loss when normally distributed (c_i are mean)
- Mean Absolute Error is alternative (c_i are median)
- For Poisson (or other distributed) data use e.g. glmtree (in R)

Trees do a recursive partitioning of the predictor space

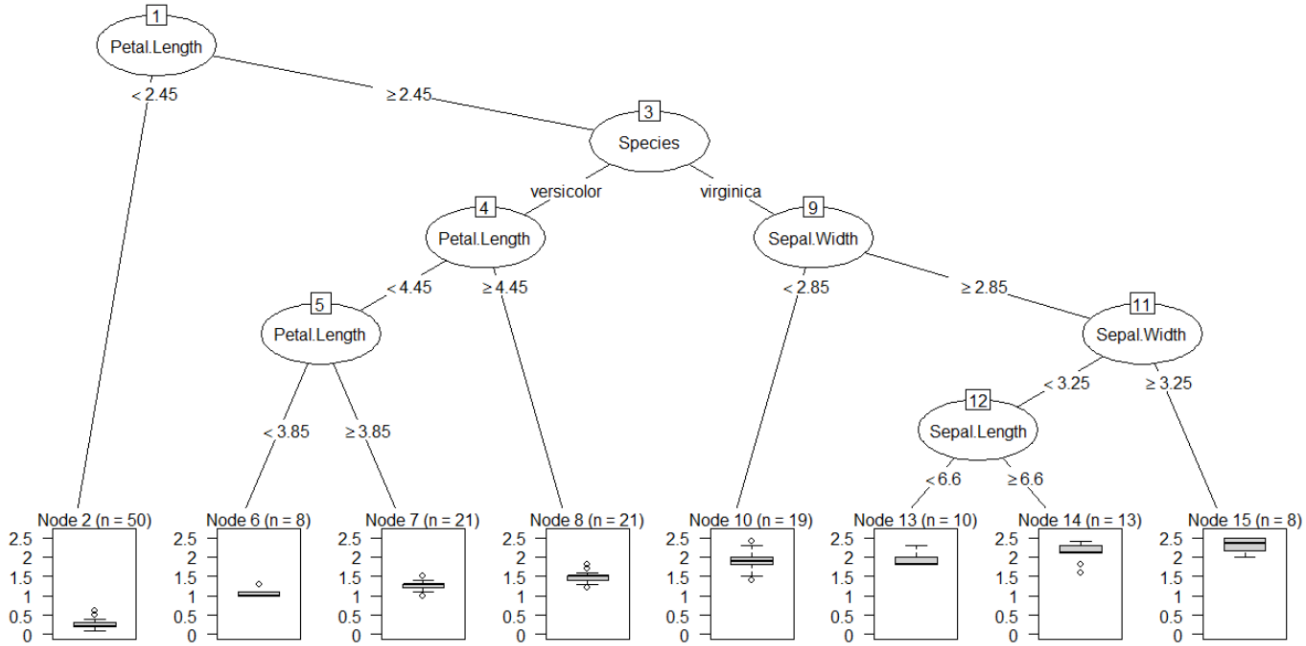






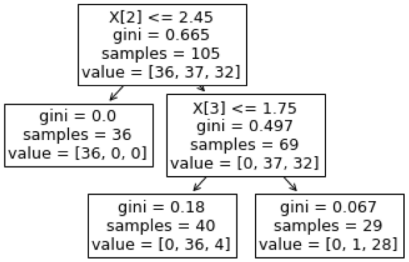
Regression trees based recursive partitioning (rpart) in R

```
library(rpart)     # to fit rpart tree models
library(party)     # only required for nice plots
library(partykit) # only required for nice plots
data("iris")
reg.tree = rpart(Petal.Width ~ ., data=iris)
plot(as.party(reg.tree))
```

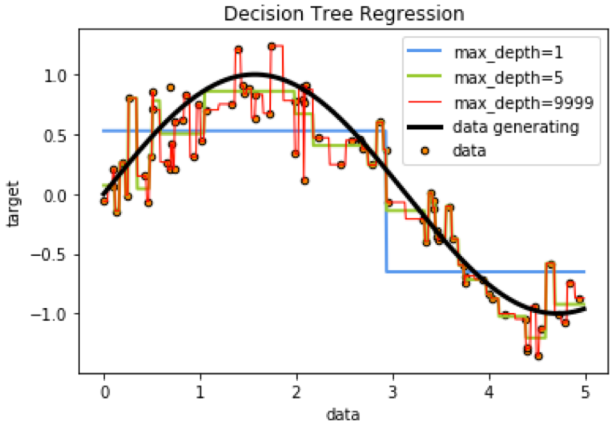


Demo

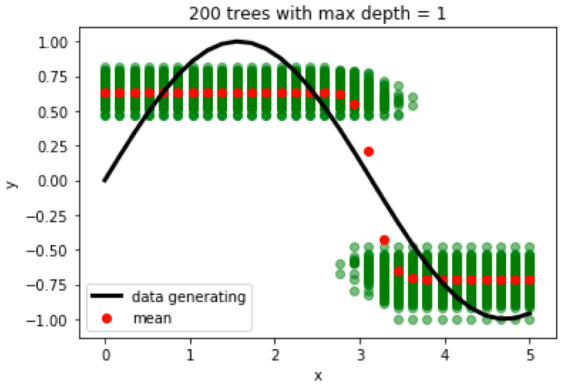
Classification

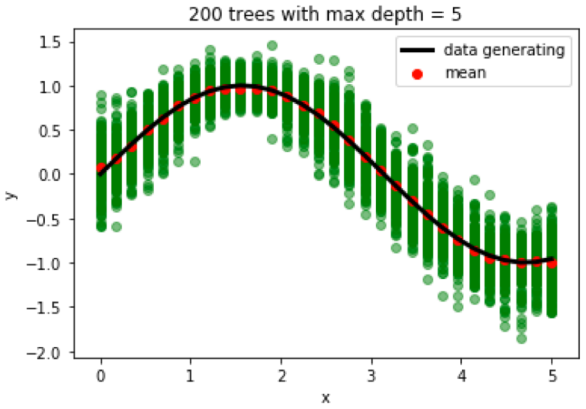


1-D regression



1-D Regression (several simulations)





- DT with low max depth have high bias
- DT with large max depth have low bias but high variance

https://github.com/ioskn/mldl_htwg/blob/master/trees_and_rf/Decission_Tree.ipynb

Pros and cons of trees

Pros

- Easy to interpret
- They can easily handle mixed discrete and continuous inputs
  - Not in scikit-learn (w.t.f.)! Use R!
- Insensitive to monotone transformations of the inputs
- Perform automatic variable selection
- Robust to outliers
- Scale to large data sets
- Multiclass and Regression

Cons

- Poor performance
- Deep Trees are unstable
  - High Variance (we will fix this …)

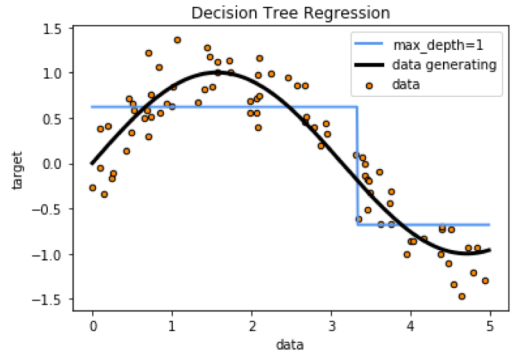
Ensemble Methods: Joining Forces

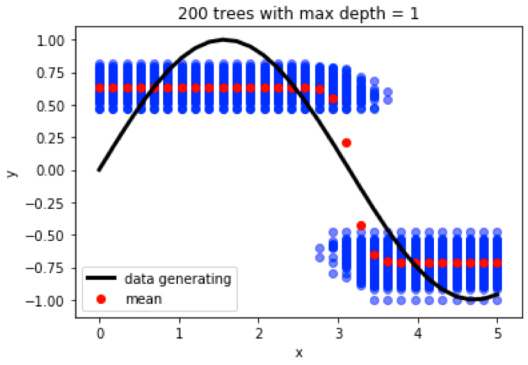
Overview of Ensemble Methods

- Many instances of the same classifier (**often trees!**)
  - Bagging (bootstrapping & aggregating)
    - Create “new” data using bootstrap
    - Train classifiers on new data
    - Average over all predictions (e.g. take majority vote)
  - Bagged Trees
    - Use bagging with decision trees
  - Random Forest
    - Bagged trees with special trick
  - (Ada) Boosting
    - An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records.
  - Gradient Boosting
    - Fits trees to residuals determined by the gradient of the loss
- Combining classifiers (different classifiers)
  - Stacking classifiers
    - Use output of classifiers as input for a new classifier

Recap: Bias Variance (Regression)

Stumps

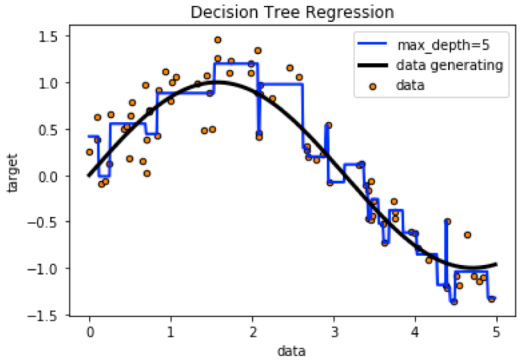


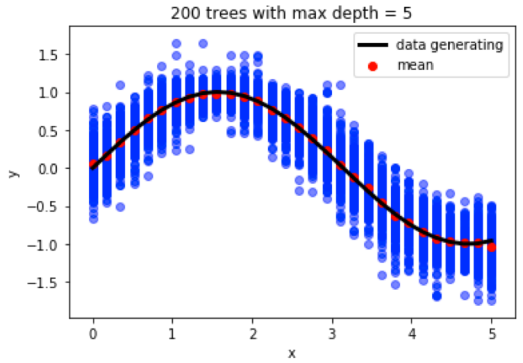


One blue dot is one tree

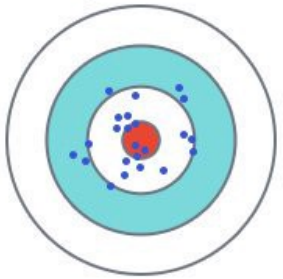
High Bias
Underfitting

Trees of depth 5





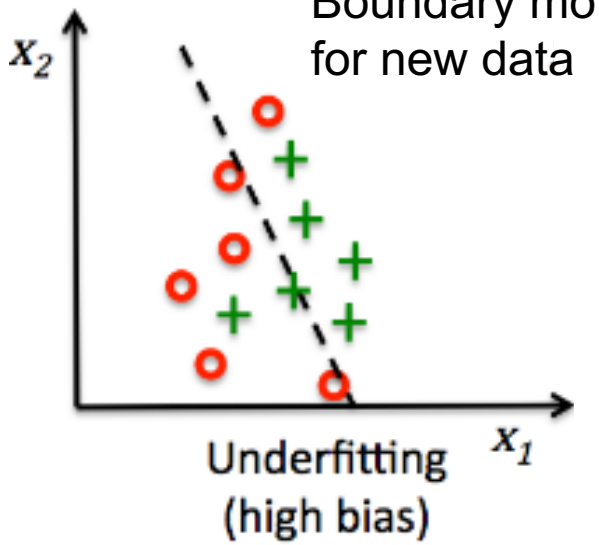
One blue dot is one tree



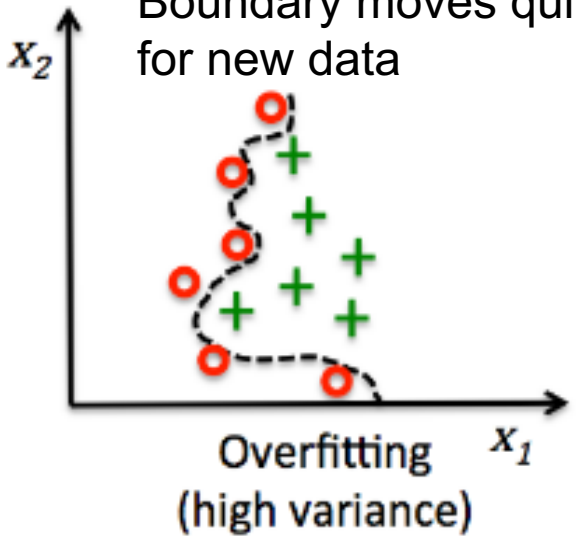
High Variance
Overfitting

Recap: Bias Variance (Classification)

Boundary moves little for new data

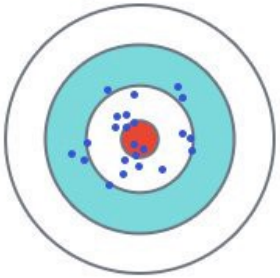


Boundary moves quite a bit for new data



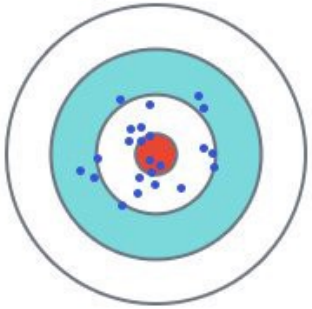
One blue dot is one classifier

One blue dot is one classifier



The concept of Bias and Variance of a classification model

One blue dot is one classifier / regressor



A underfitting model

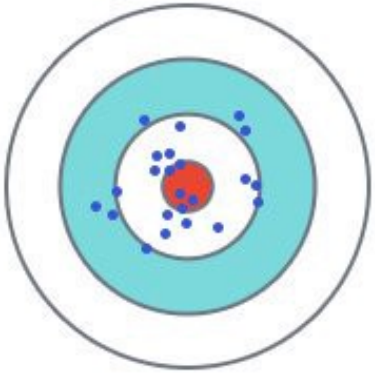
- is not flexible enough
- quite many errors on train data and **systematic test error (high bias)**
- **will not vary much if new train data is sampled from population (low variance)**

A overfitting model

- is too flexible for data structure
- few errors on train set and non-systematic test errors **(low bias)**
- will vary a lot if fitted to new train data **(high variance)**

Use ensemble methods to fight under and overfitting

High Variance



High Bias

fight the deficits of the single model by

Bagging

improve ensemble approach further by

Random Forest
in case of tree models

Adaptive boosting

Gradient boosting
(fights under- and overfitting)

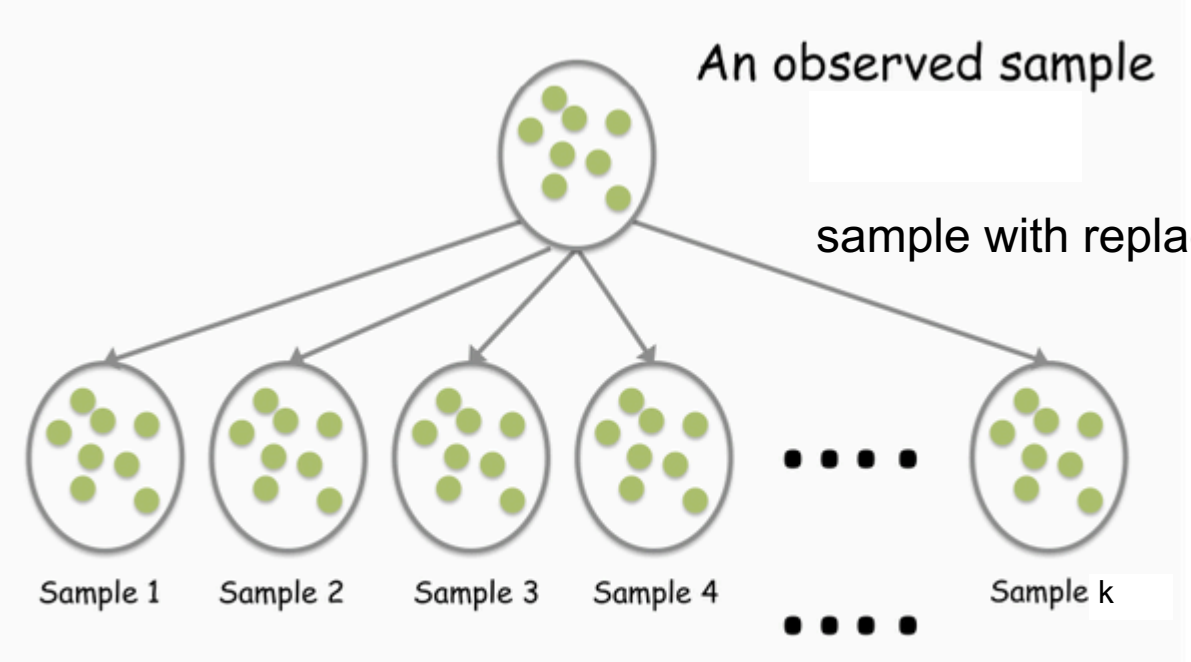
Ensemble methods are the cure!

http://www.spiegel.de/spam/humor-fuer-leute-mit-humor-nel-ueber-schwarmintelligenz-a-842671.html

Bagging
&
Random Forest

Principle of Bootstrapping

- How to create new data (if you can't simulate)?



Picture taken from https://github.com/founddrama/

Bagging as ensemble of parallel fitted models

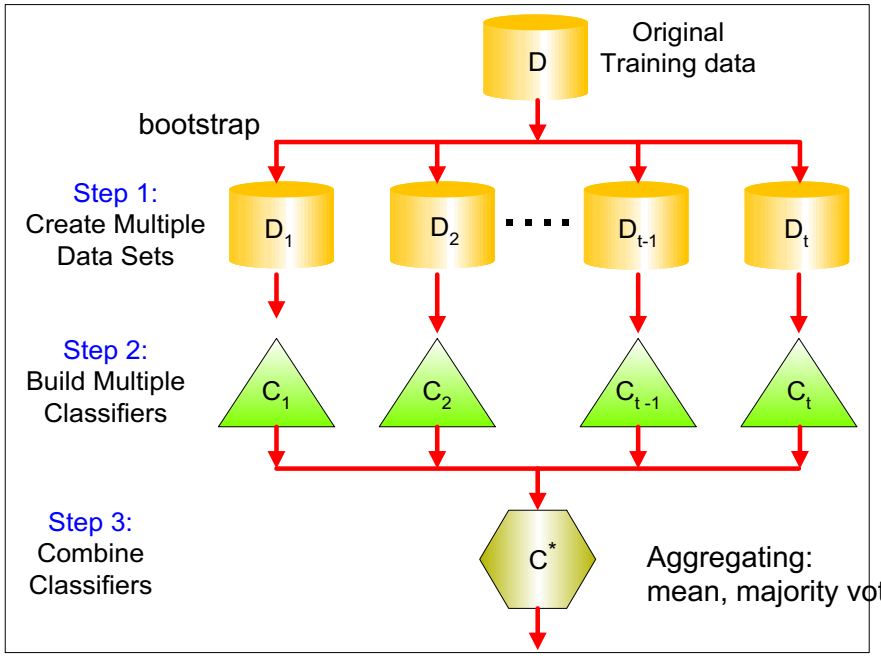
Each classifier tends to overfit its version of training data.

Bagging: bootstrapping and averaging

1) Fit flexible models on different bootstrap samples of train data

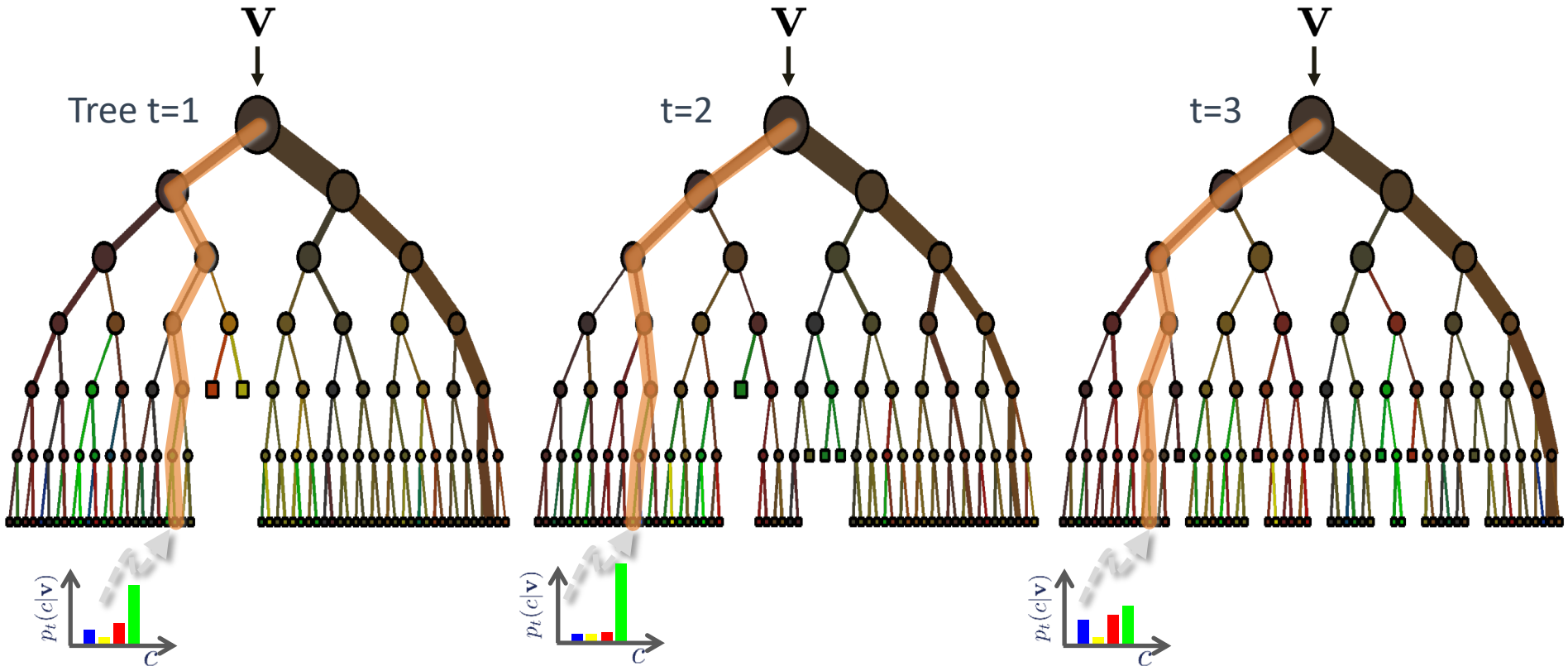
2) Minimize Bias by using flexible models and allow for overfitting

3) Reduce variance by averaging over many models



Remarks: highly non-linear estimators like trees benefit the most by bagging
If model does not overfit the data bagging does not help nor hurt.

How to classify a new observation v with a random forest?

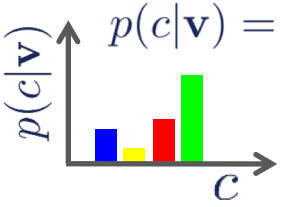


The to derive the ensemble result:

a) Each tree has a “winner-class”
Take the class which was most often the winner

b) average probabilities:

$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$$



Animation taken from microsoft

Why does it work?

- Suppose there are 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
- Assume classifiers are independent (that's the most questionable assumption)
- Take **majority vote**
- Majority voter is wrong if 13 or more are wrong
- **Number of wrong predictors???**

Why does it work [Solution]

- Suppose there are 25 base classifiers
- Each classifier has error rate, ε = 0.35
- Assume classifiers are independent (that's the most questionable assumption)
- Take **majority vote**
- Majority voter is wrong if 13 or more are wrong
- Number of wrong predictors X ~ Bin(size=25, p_0=0.35)

$$\sum_{i=13}^{25}\binom{25}{i}\varepsilon^i(1-\varepsilon)^{25-i} = 0.06$$

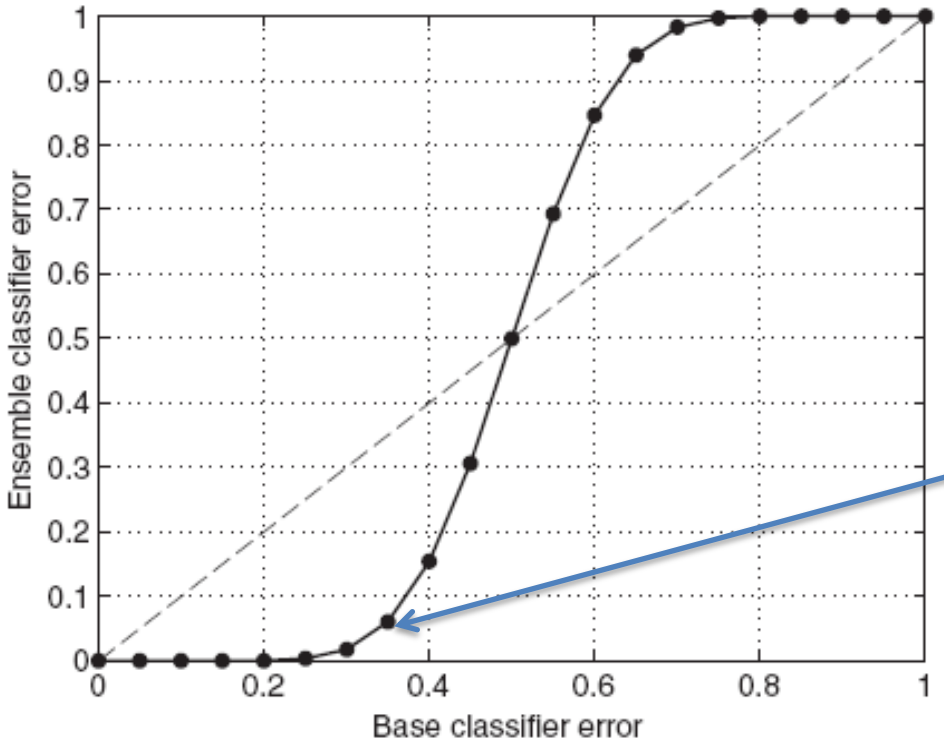
> 1 - pbinom(12, size=25, prob = 0.35) or in python
[1] 0.06044491

```
1 - scipy.stats.binom.cdf(12, 25, 0.35)
```

```
0.06044491356702042
```

Why does it work?

- 25 Base Classifiers

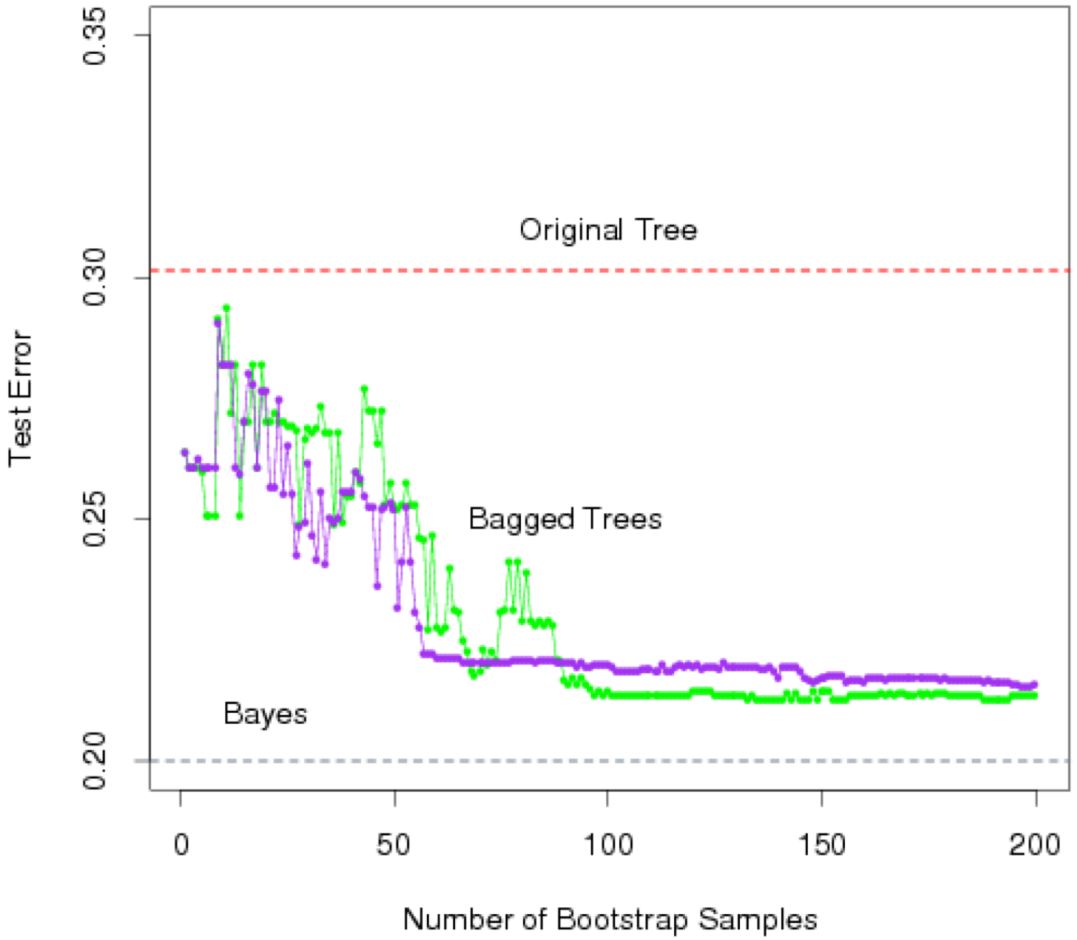


See last slide
1 - pbinom(12, size=25, prob = 0.35)

Ensembles are only better than one classifier, if each classifier is better than random guessing! Assuming independence between the classifiers.

A Comparison of Error Rates

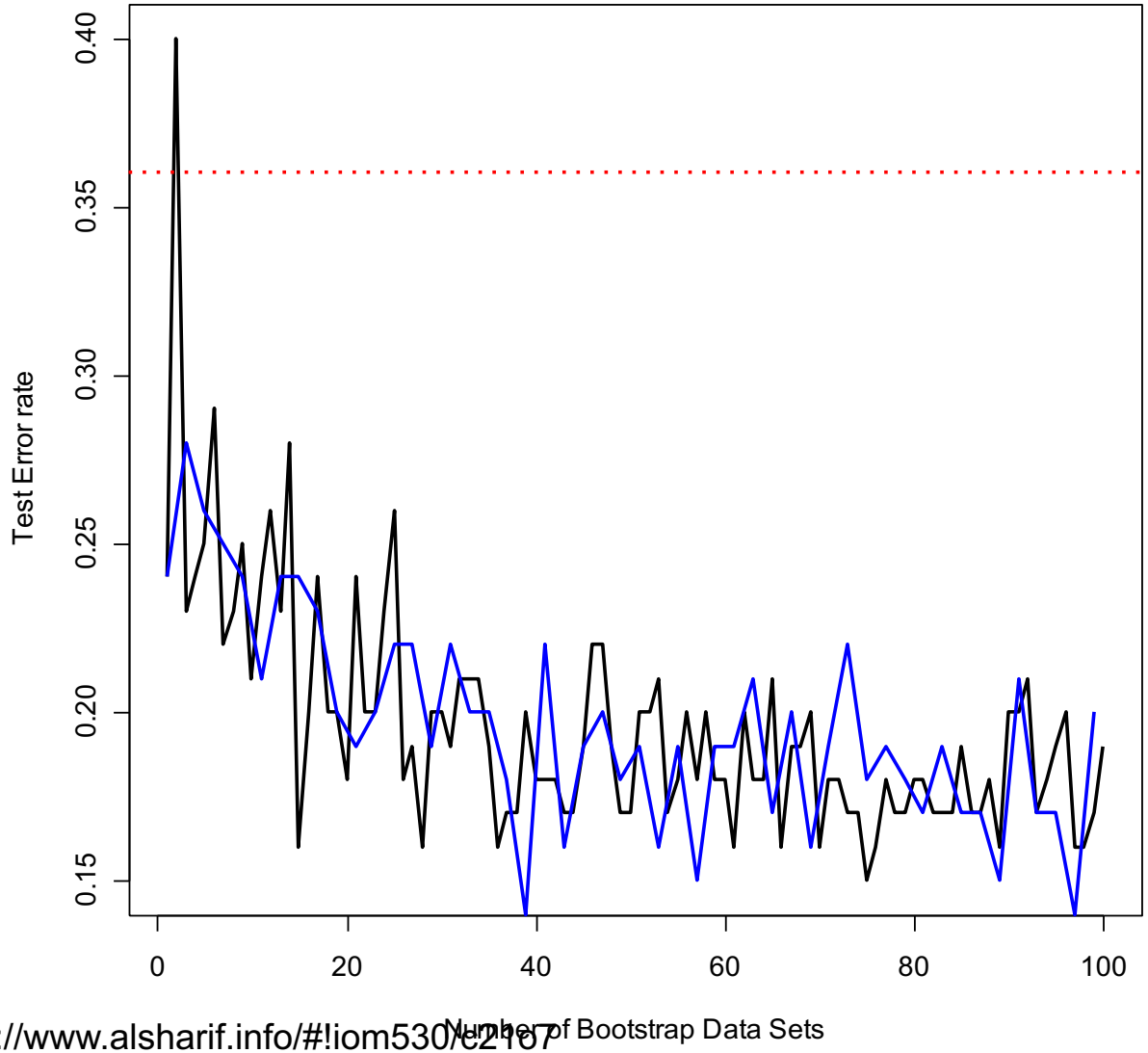
- Here the green line represents a simple majority vote approach
- The purple line corresponds to averaging the probability estimates.
- Both do far better than a single tree (dashed red) and get close to the Bayes error rate (dashed grey).



Slide taken from Al Sharif: http://www.alsharif.info/#!iom530/c21o7

Example 2: Car Seat Data

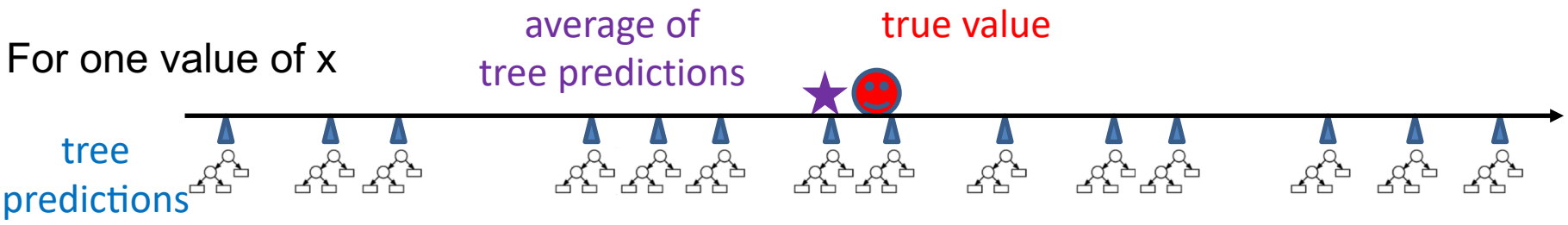
- The red line represents the test error rate using a single tree.
- The black line corresponds to the bagging error rate using majority vote while the blue line averages the probabilities.



Slide taken from Al Sharif: http://www.alsharif.info/#!iom530/c2167

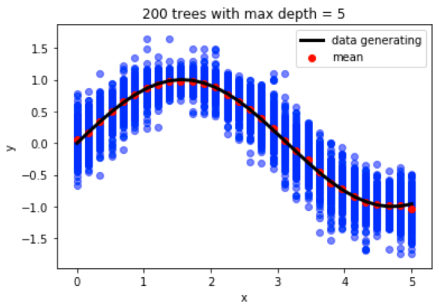
Aggregate of regression trees?

- Aggregation by taking the mean
- Suppose there are n=25 deep regression trees
  - All trees are deep -> the trees have no or only small bias
  - All trees are deep -> the trees have high variance
  - Handwaving: According to the Central Limit Theorem the average of the predictions of n regression trees have the same expected value as the single trees but a standard deviation which is reduced by a factor of $\sqrt{n} = \sqrt{25} = 5$.



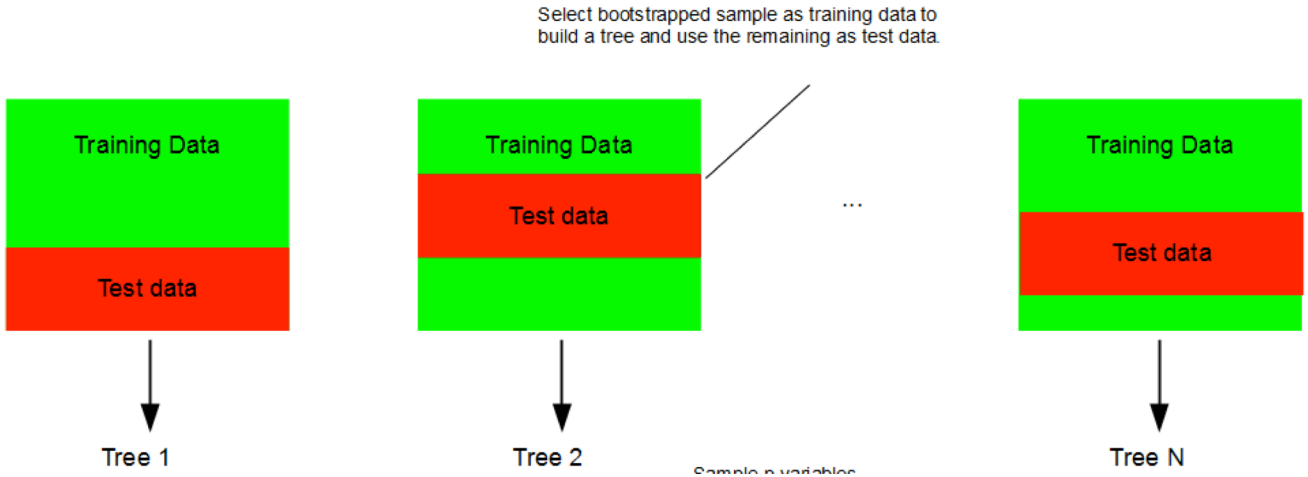
predictions for the same observation
made by different bootstrap models

1-D Example
30 values of X



Out-of-Bag Estimation: "Cross-validation on the fly"

- Since bootstrapping involves random selection of subsets of observations to build a training data set, then the remaining non-selected part could be the testing data.
- On average, each bagged tree makes use of around 2/3 of the observations, so we end up having 1/3 of the observations used for testing



Slide taken from Al Sharif: http://www.alsharif.info/#!iom530/c21o7

The importance of independence

- It's a quite strong assumption, that all classifiers are independent.

- Since the same features are used in each bag, the resulting classifiers are quite dependent.
  - **We don't really get new (uncorrelated) classifiers or trees**

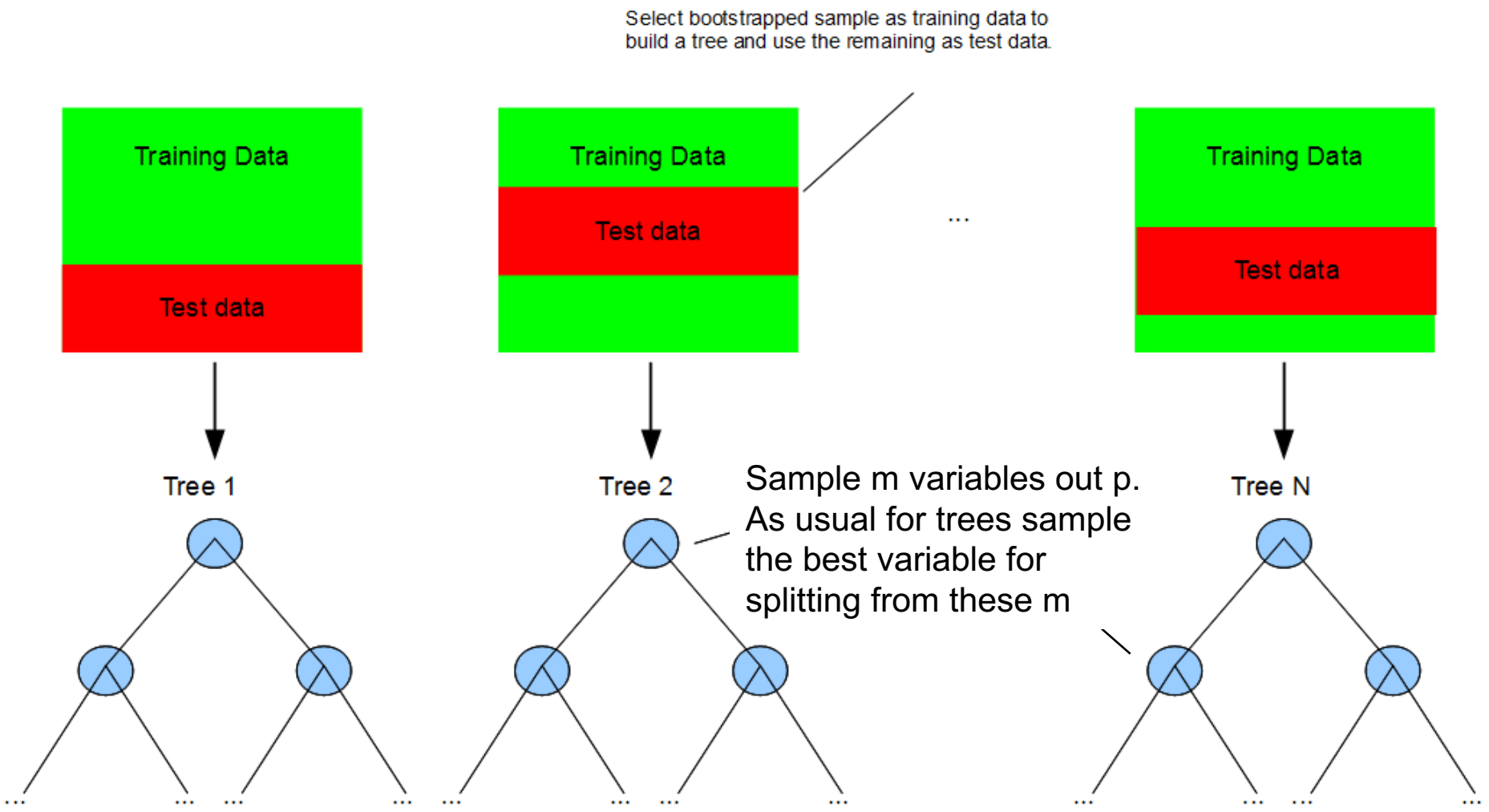
- **De-correlate by any means! ➔ Random Forest**
  - We are willing that the individual trees have worse performance
  - Our main goal is to decorrelate, we are quite brutal…

Random Forest

Random Forests

- Very efficient statistical learning method
- It builds on the idea of bagging, but it provides an improvement because it de-correlates the trees
  - Different trees are independent
- How does it work?
  - Build a number of decision trees on bootstrapped training sample, but when building these trees, each time a split in a tree is considered, **a random sample of *m* features** is chosen as split candidates from the full set of *p* features (usually $m \approx \sqrt{p}$ for classification)

Random Forest: Learning algorithm



Random Forest with different values of "m"

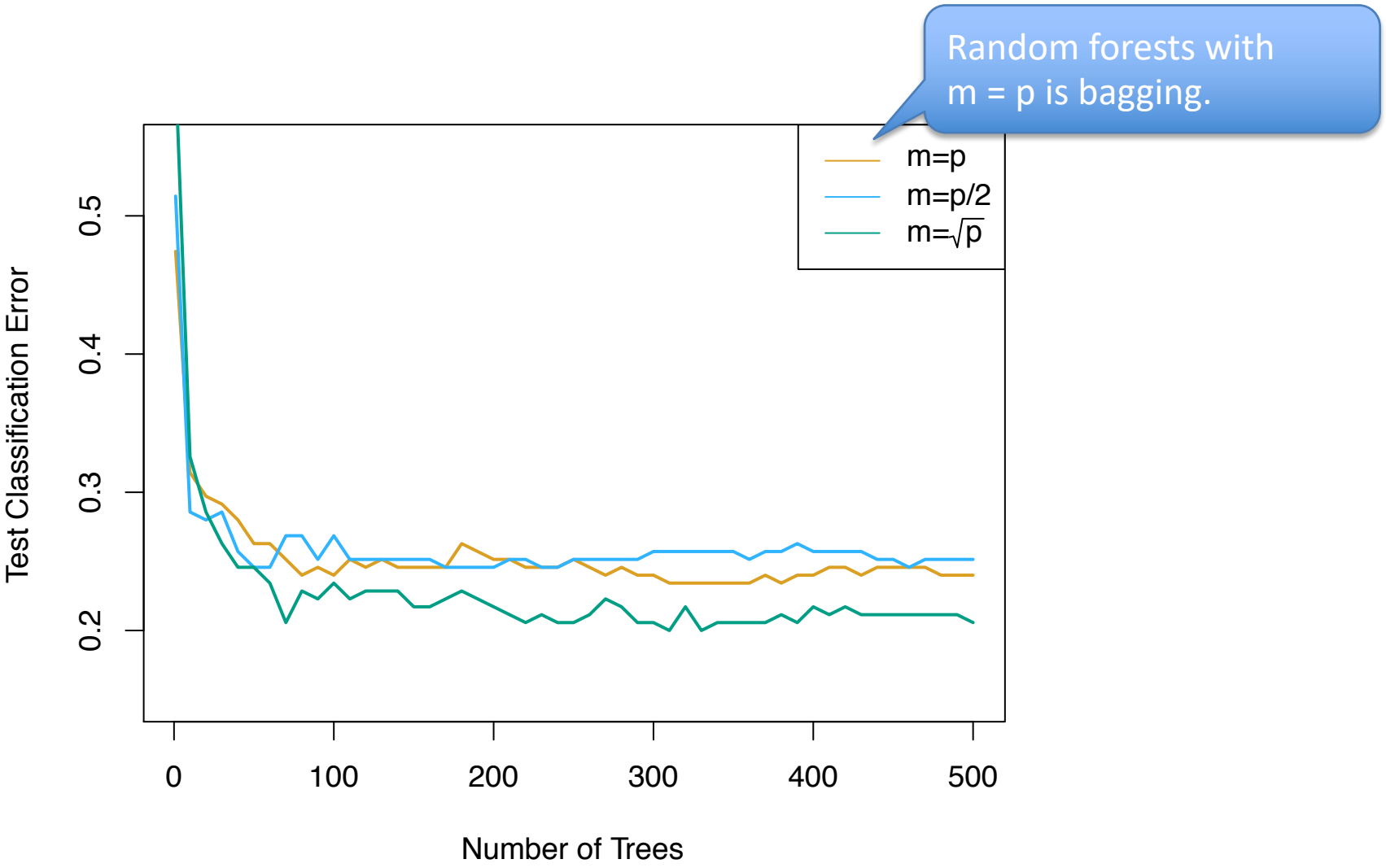


Figure from ISLR

Random Forest in sklearn

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, oob_score=True)
rf.fit(X,y)
rf.oob_score_


# max_features='auto’#max_features corresponds to m
```

Variable Importance Measure

- Bagging (and RF) typically improves the accuracy over prediction using a single tree, but it is now hard to interpret the model!
- We have hundreds of trees, and it is no longer clear which variables are most important to the procedure
- Thus bagging improves prediction accuracy at the expense of interpretability
- But, we can still get an overall summary of the importance of each predictor using relative influence plots

Slide taken from Al Sharif: http://www.alsharif.info/#!iom530/c21o7

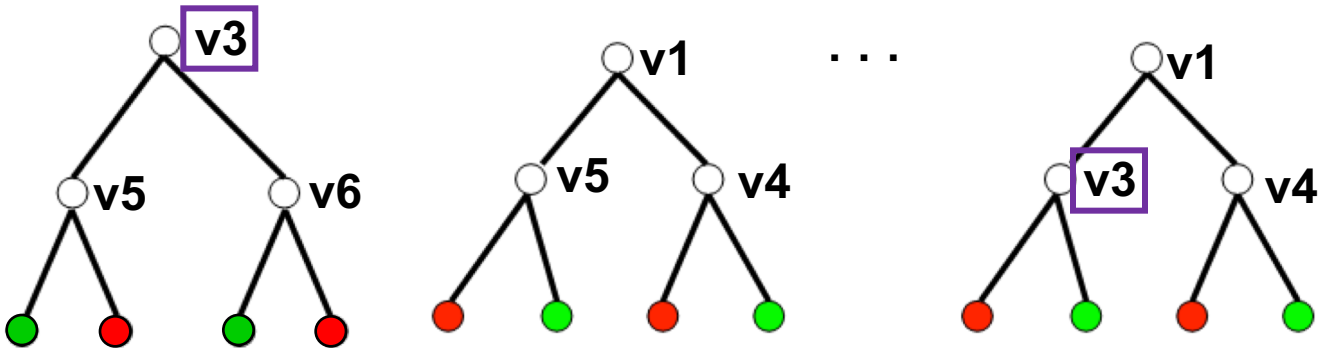
Variable Importance 1: performance-loss by permutation

Determine importance-2 for v4:

1) obtain standard OOB-performance
2) values of v4 are randomly permuted in OOB samples and OOB-performance is again computed. If variable is not important nothing happens. Hence:
3) Use decrease of performance as importance measure of v4

v1	v2	v3	v4	v5	v6	v7
0	1	1	2	0	1	0
0	2	2	1	2	0	1
1	0	0	1	1	2	0
1	0	0	1	1	0	2
0	2	1	0	2	0	1

Variable Importance 2: Score improvement at each Split



At each split where the variable, e.g. v3, is used the improvement of the score (Decrease of Giniindex) is measured. The average over all these v3-involving splits in all trees is the “MeanDecreaseGini” measure of v3.

Known Pros of Random Forest

Pros

- It is very easy to handle since based on Trees
  - No transformation needed
  - Can handle categorical features (and a mixture of categorical and numerical features)
  - Can handle missing values
- Random Forest does not tend to overfit.
- CV incorporated OOB
  - Observations must be independent (Otherwise same observations in ”test and Train”
- Can handle lots of noise variable even with quite few relevant variables (6 out of 100 already works fine)
  - No variable selection needed (in general)
- Importance of variables can be quantified -> variable importance
- It can compute proximities between observations -> clustering data [not shown here]

Cons

- Quite accurate classifier, but usually not the best.
- **Ideal baseline**
- Hard to interpret

Let's praise the Random Forest

RF is not in all situations the best method - however it often just works!

The Random Forest™ is my shepherd; I shall not want.
He makes me watch the mean squared error decrease rapidly.
He leads me beside classification problems.
He restores my soul.
He leads me in paths of the power of ensembles
for his name's sake.

Even though I walk through the valley of the curse of dimensionality,
I will fear no overfitting,
for you are with me;
your bootstrap and your randomness,
they comfort me.

You prepare a prediction before me
in the presence of complex interactions;
you anoint me data scientist;
my wallet overflows.
Surely goodness of fit and money shall follow me
all the days of my life,
and I shall use Random Forests™
forever.

Christoph Molnar http://www.statsblogs.com/2014/02/06/random-forest-almighty/

Adaptive Boosting

Historical View on boosting methods

- Adaptive Boosting (adaBoost) *Algorithm*
  - Freund & Schapire, 1990-1997
  - An *algorithmic* method based on iterative data reweighting for *two class* classification
- Gradient Boosting (GB)

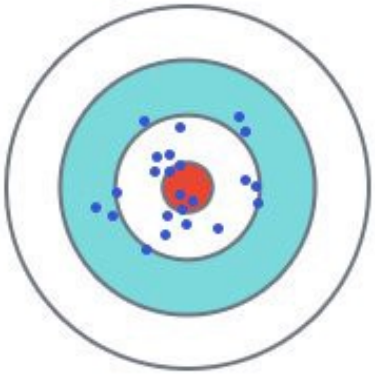
  "There is nothing more practical then a good theory" (Kurt Lewin)
  - Breiman (1999) Understood that bosting can be understood as an optimization of a loss function (opening the door to > 2 class classification and regression)…
  - Friedman/Hastie/Tibshirani (2000) "Gradient Boosting Machines"
    - Generalization to a variety of loss functions

- New powerful implementations / extensions (starting ~2014)
  - Extreme Gradient Boosting (xgboost - a particular implementation of GB)
    - Tianqi Chen (2014 code, 2016 published arXiv://1603.02754)
    - Often used in Kaggle Competitions as part of the winning solution
  - CatBoost (Yandex arXiv://1706.09516) with categorical data and on GPU

- Still active research e.g.
  - "Transformation Boosting Machines" (T. Hothorn 2019)

Use ensemble methods to fight under and overfitting

High Variance



fight the deficits of the single model by

Bagging

improve ensemble approach further by

Random Forest
in case of tree models

High Bias

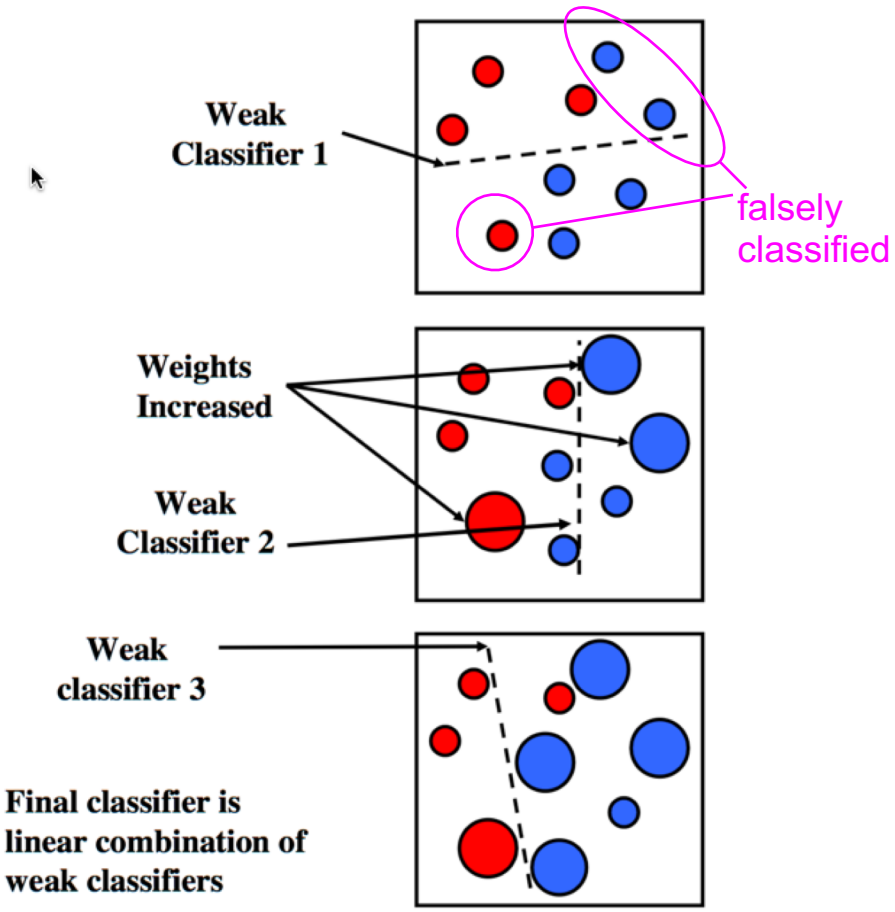
Adaptive boosting

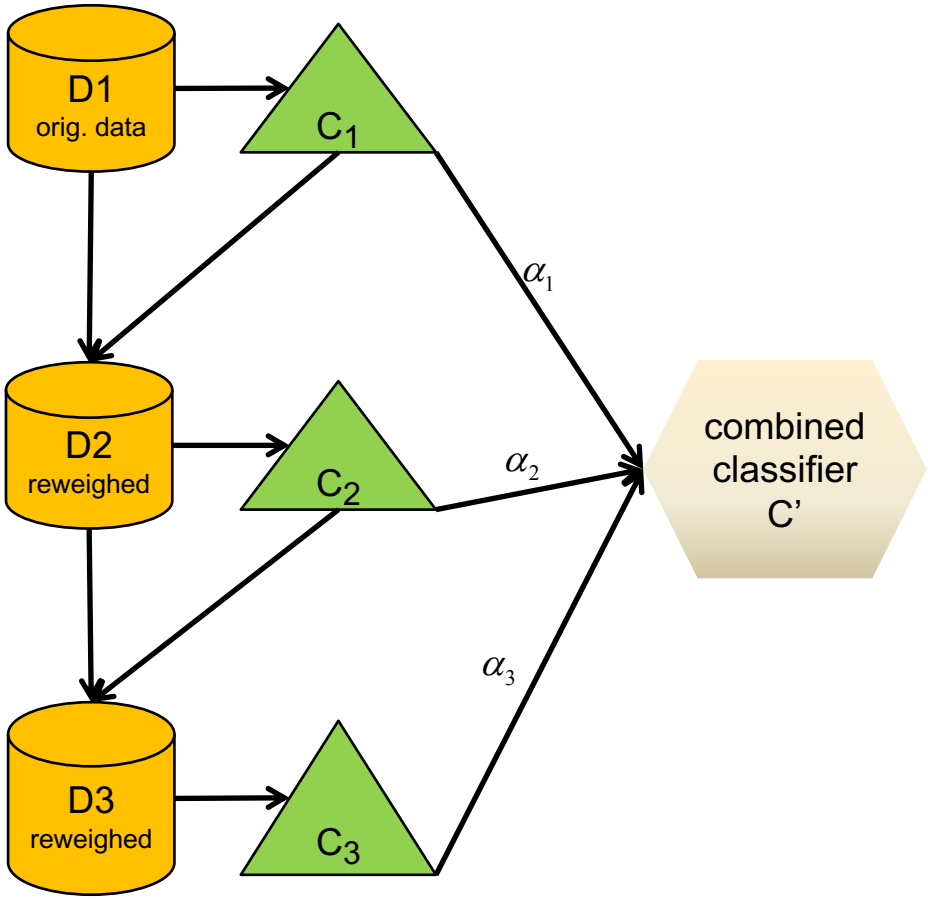
Gradient boosting
(fights under- and overfitting)

Adaptive Boosting as ensemble of sequentialy fitted models

We use in each step a simple (underfiting) model to fit the current version of the data.

After each step those observations get up-weighted, that were misclassified (two classes)

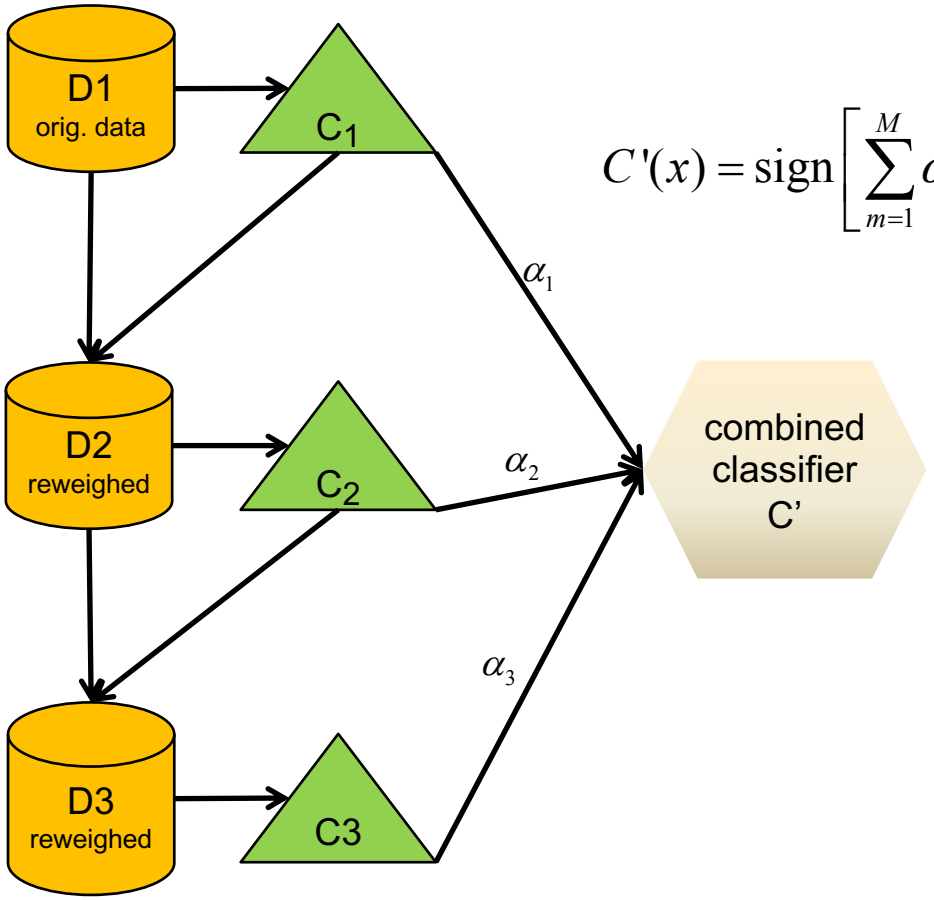




left fig credicts:
http://vinsol.com/blog/2016/06/28/computer-vision-face-detection/

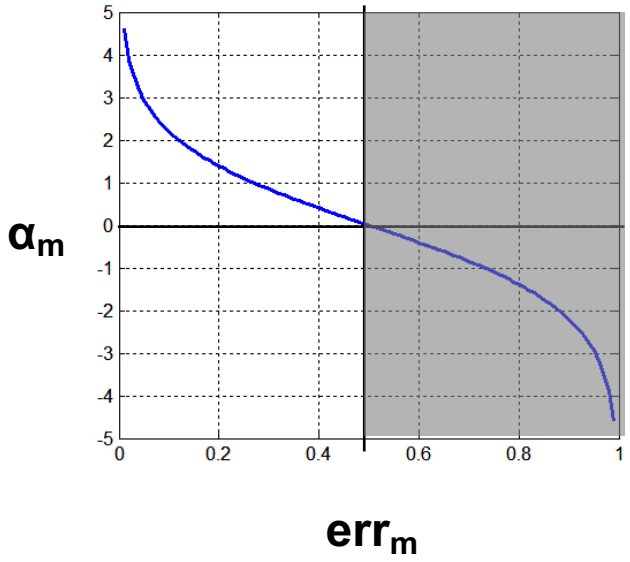
Adaptive Boosting as weighted average of sequential models

The final classifier C' is a weighted average of all sequential models C_m, where the model-weights α_m are given by the misclassification rate err_m of the model C_m taking into account the observation-weights of the used reweighted data set D_m.



$$C'(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m \cdot C_m(x)\right]$$

C_m error small ➔large weight α_m



Details of Ada Boost Algorithm

Algorithm:

1) Set $y_i \in \{-1,+1\}$ and start with identical weights $w_i = 1/n$

2) Repeat for $m = 1,2,...,M$:

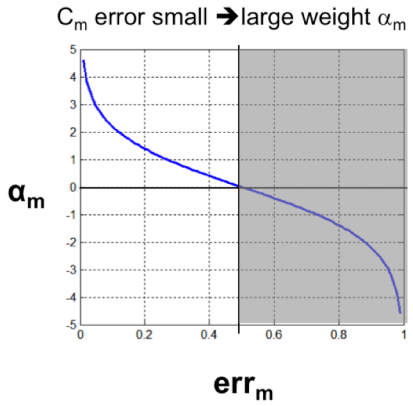
   a) Fit the classifier $f_m(x) \in \{-1,+1\}$ using weights w_i

   b) Compute the weighted error $err_m = \sum_i w_i \cdot I[y_i \neq f_m(x_i)]$

   c) Compute the aggregation weight $\alpha_m = \log((1-err_m)/err_m)$

   d) Set $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot I[y_i \neq f_m(x_i)])$; normalize to $\sum_i w_i = 1$

3) Output $F_M(x) = sign\sum_{m=1}^{M} \alpha_m f_m(x)$



Remark: One can show (see ELS chapter 10.4, p.343) that the reweighting algorithm of AdaBoost is equivalent to optimizing an exponential loss.

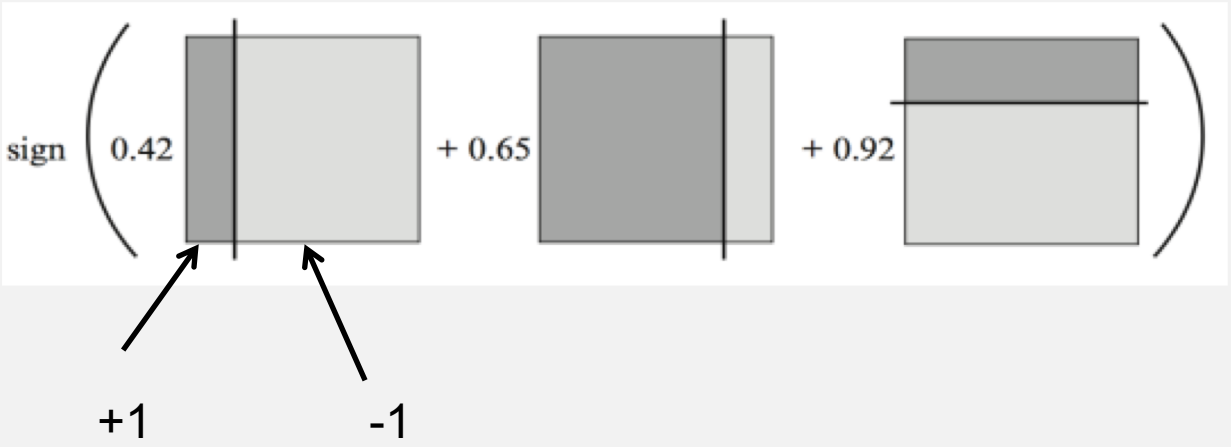
Ada Boost in simple words

- Fit an additive model $\sum_{m=1}^{M} \alpha_m \cdot C_m$ (ensemble) in a forward stage-wise manner.
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- In Adaboost,“shortcomings” are identified by high-weight data points.
- Weak learns often trees, sometimes only grown to depth 1 (stumps)

Example of such a boosted tree with 3 stumps…

Example (You Turn)

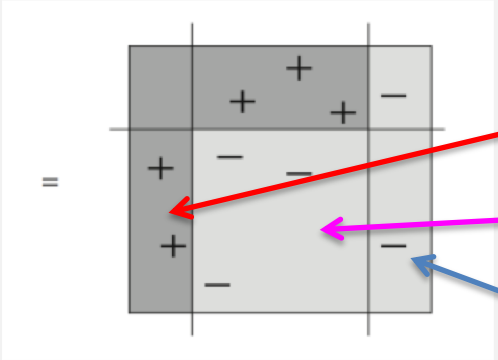
$$F_3(x) = \text{sign}\left(\sum_{m=1}^{3} \alpha_m f_m(x) \right)$$



Example (You Turn)

Lösung

$$F_3(x) = \text{sign}\left(\sum_{m=1}^{3} \alpha_m f_m(x) \right)$$



> 0.4+0.65-0.92
[1] 0.13 # +
> -0.42+0.65-0.92
[1] -0.69 # -
> -0.42-0.65-0.92
[1] -1.99 # -

Performance of Boosting / Diagnostic Setting

Look at a specific case

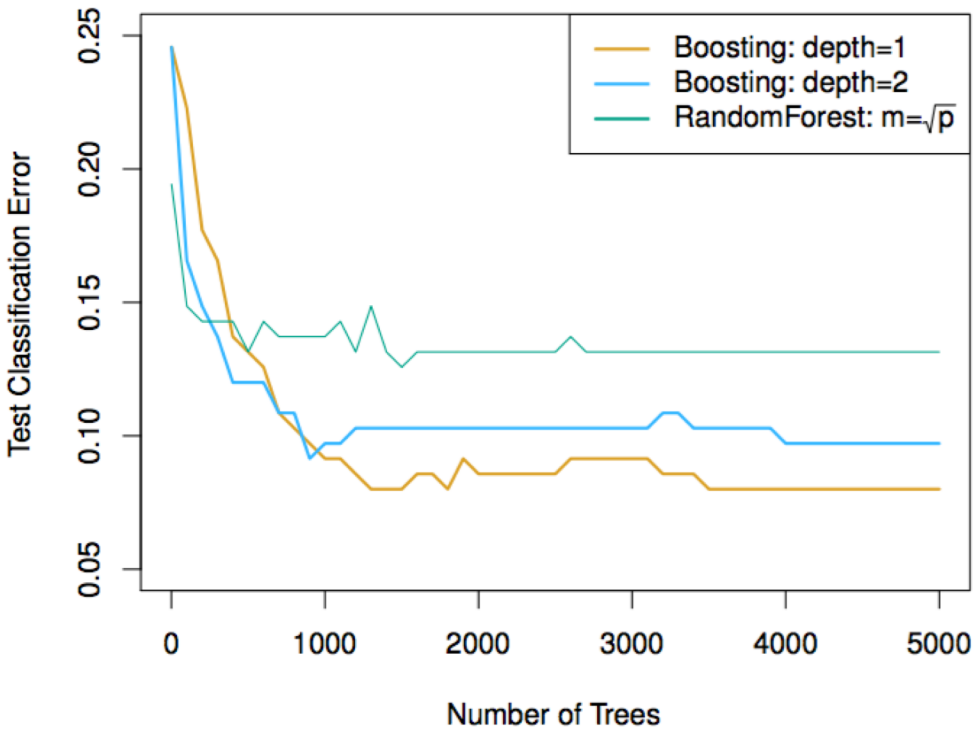


FIGURE 8.11 (ISLR) Results from performing boosting and random forests on the 15-class gene expression data set in order to predict cancer versus normal

Boosting most frequently used with trees (not necessary).

Trees are typically only grown to a certain depth – often 1 or 2.

Diagnostic: Significant interaction if depth 2 works better than depth 1.

Gradient Boosting

We follow the line of thought of “the elements of statistical learning” (ESL) chapter 10

Content

- For MSE
  - Boosting is stagewise additive fitting of the residuals
  - Overfitting: early stopping and learning rate
- Generalization
  - Generalization of residuals are negative gradients of a loss function
- Different Loss Functions

Boosting fits an additive model

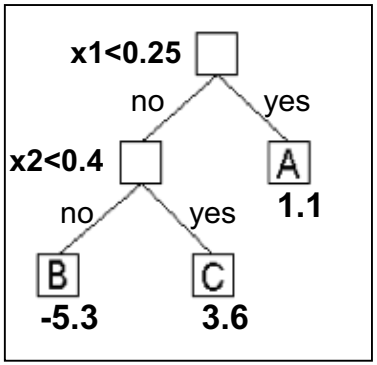
- Let's look at the decision function of Adaboost

$$F_M(x) = sign\underbrace{\sum_{m=1}^{M} \alpha_m f_m(x)}_{f(x)}$$

- This can be seen as the signum of

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

  - Where β are expansion coefficients of (simple) basis functions b(x)
  - Function expansions are found in many places
    - Wavelets γ_m location and scale shifts of a mother wavelet
    - …
    - Here γ_m and β_m parameterize the tree (split point and predictions at the leaves).



Forward stagewise additive modeling

- Minimizing a loss function (MSE or NLL) on the training data $\{y_i, x_i\}_{i=1,\ldots,N}$

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right)$$

← Too complex to fit all at once.

- Approximation: step by step (stagewise)

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)).$$

old model

new model

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)$.

Comparison between binary crossentropy and exponential

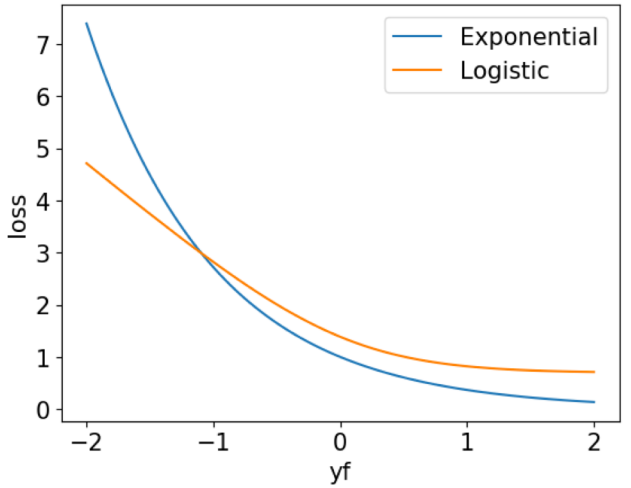
- Adaboost

$$F_M(x) = sign\sum_{m=1}^{M}\alpha_m f_m(x)$$

Corresponds to the following loss function (per training example *i*)

  - $\text{loss} = e^{-f(x_i)\cdot y_i}$ remember $y_i \in \{-1, +1\}$
  - See ELS 10.5

- Alternative: use logistic regression
  - Logistic regression is better (more stable)
  - See ELS 10.6



Example: Least Squares Regression $L(y, f(x)) = (y - f(x))^2$

- After $m-1$ steps, suppose we have the model $f_{m-1}(x) = \sum_{j=1}^{m-1} \beta_j b(x; \gamma_j)$. ← sum of m-1 trees*
- At the mth step we solve

$$\min_{\beta,\gamma} \sum_{i=1}^{N} \left(y_i - \overbrace{f_{m-1}(x_i) - \beta b(x_i;\gamma)}^{f_m(x_i)}\right)^2$$

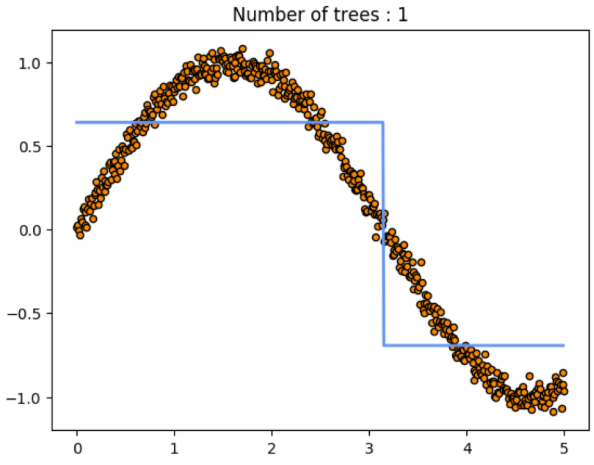
residual!

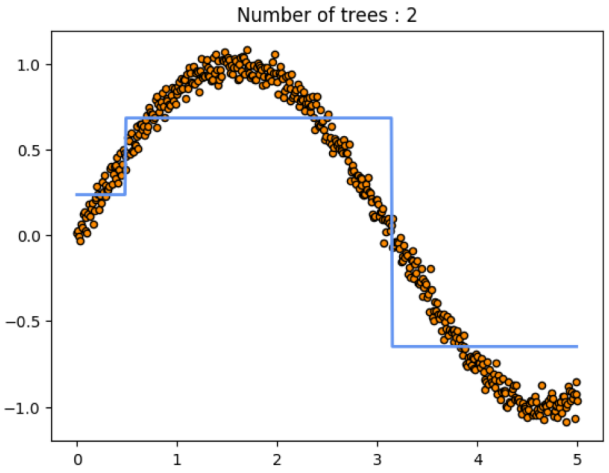
← Fitting a new tree m

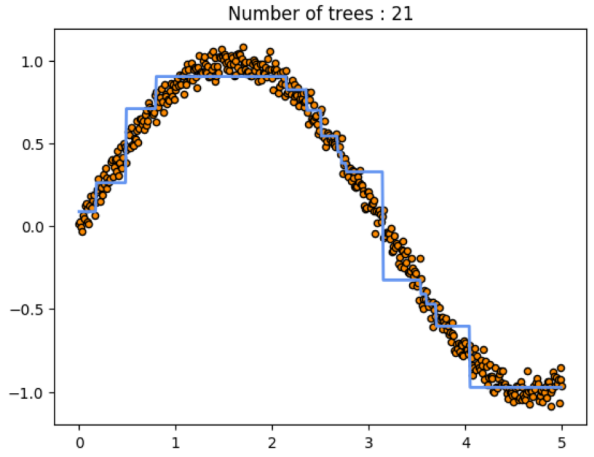
Instead of fitting a tree to the data ($f_o(x) = 0$), we fit a tree to the residuals of the previous trees.

Demo-Time: Least Square Regression

```
fx = np.zeros(num_data) #See Step 1 in Algo 10.2
for m in range(9):
    tree = DecisionTreeRegressor(max_depth=1)
    res = y - fx #Calculation of the rediduals
    tree.fit(X, res) #Fitting to the residuals (Steps 2a)
    fx = fx + tree.predict(X) #The new predictions (Step 2b)
    plot_current_f(X,y,fx,m+1)
    plt.show()
```

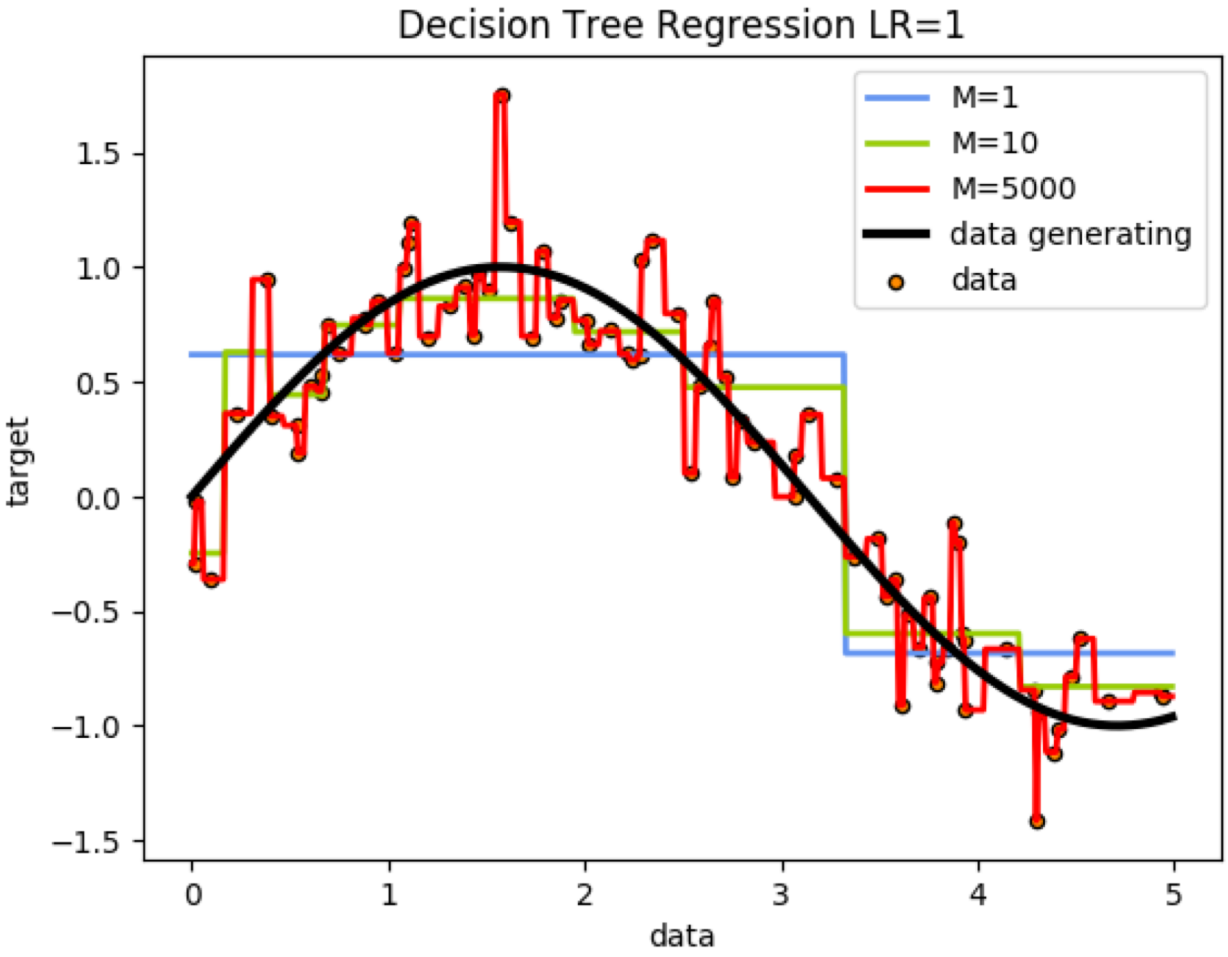






https://github.com/ioskn/mldl_htwg/blob/master/boosting/Least_Square_Boosting.ipynb

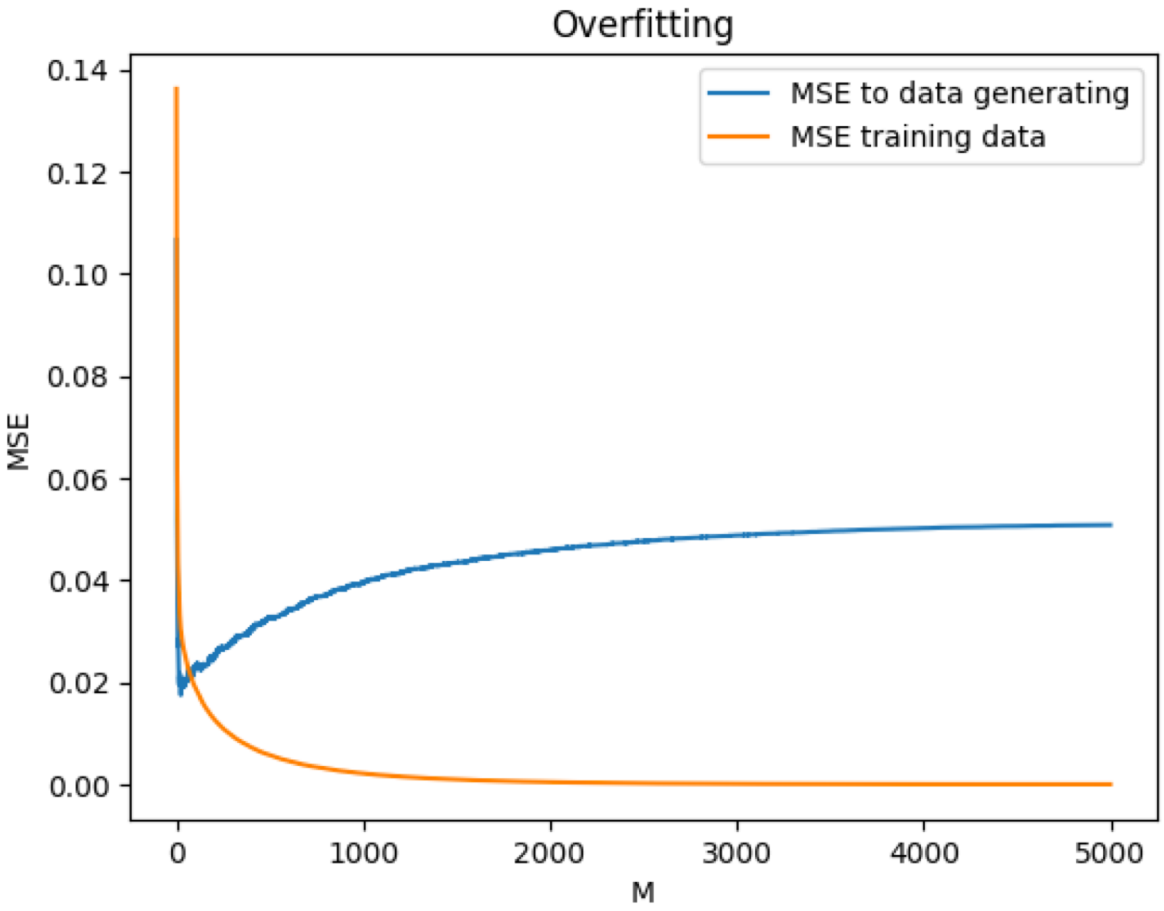
More examples



Quite greedy! Shows overfitting if too many trees are used.

https://github.com/ioskn/mldl_htwg/blob/master/boosting/Least_Square_Boosting.ipynb

Overfitting: Early Stopping



Use validation set to find the optimal number of trees M (*"early stopping"*)

https://github.com/ioskn/mldl_htwg/blob/master/boosting/Least_Square_Boosting.ipynb

Learning rate

- Scale the contribution of the new tree, for regularization

```
fx = np.zeros(num_data) #See Step 1 in Algo 10.2
for m in range(1000):
    tree = DecisionTreeRegressor(max_depth=1)
    res = y - fx #Calculation of the rediduals
    tree.fit(X, res) #Fitting to the residuals (Steps 2a)
    fx = fx + LR * tree.predict(X) #The new predictions (Step 2b) with LR
```

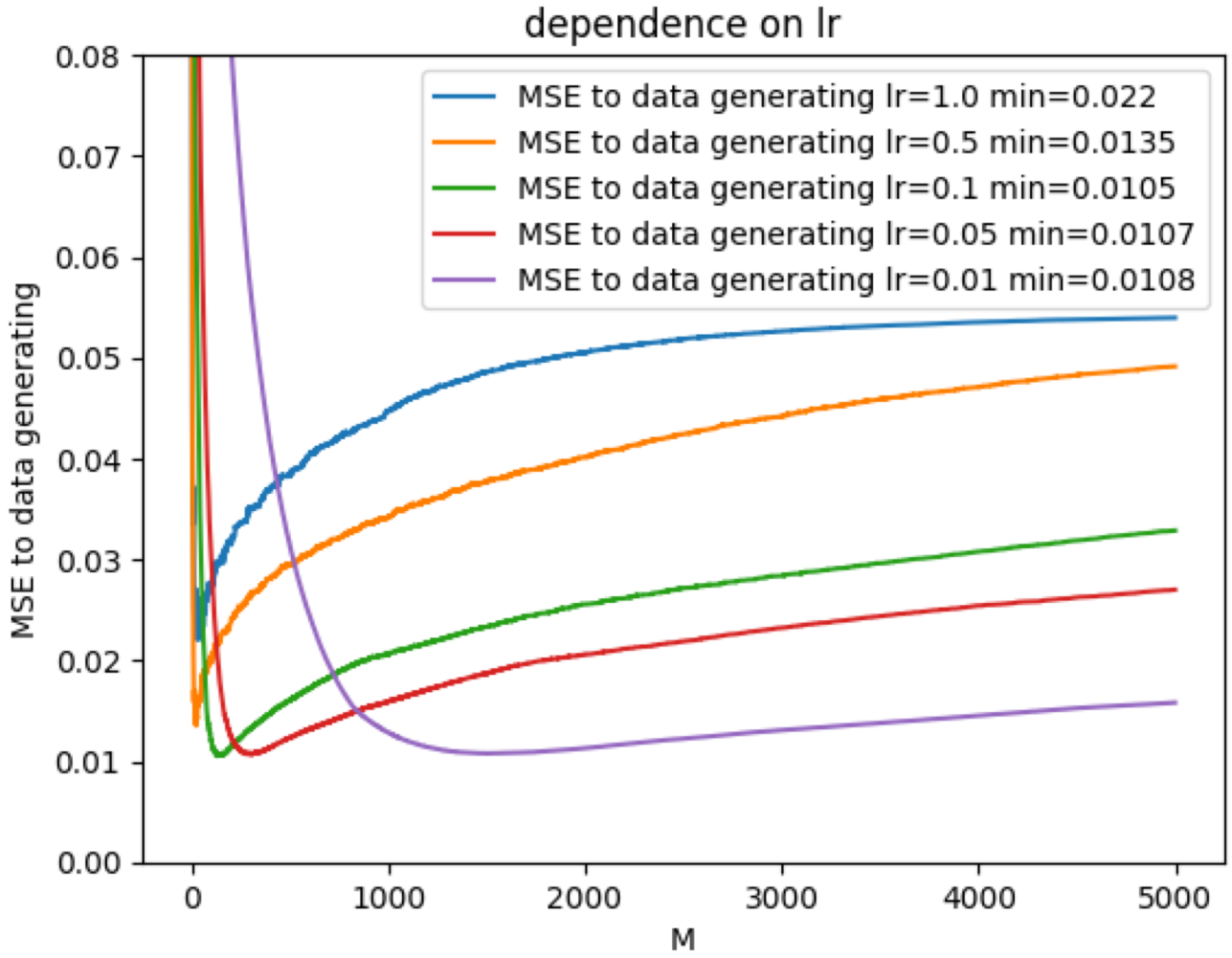
LR a new hyper parameter a.k.a. `learning_rate`, η, ν

sklearn implementation

```
from sklearn.ensemble import GradientBoostingRegressor
est = GradientBoostingRegressor(n_estimators=1000, learning_rate=LR,
                                max_depth=1, random_state=0, loss='ls')
```

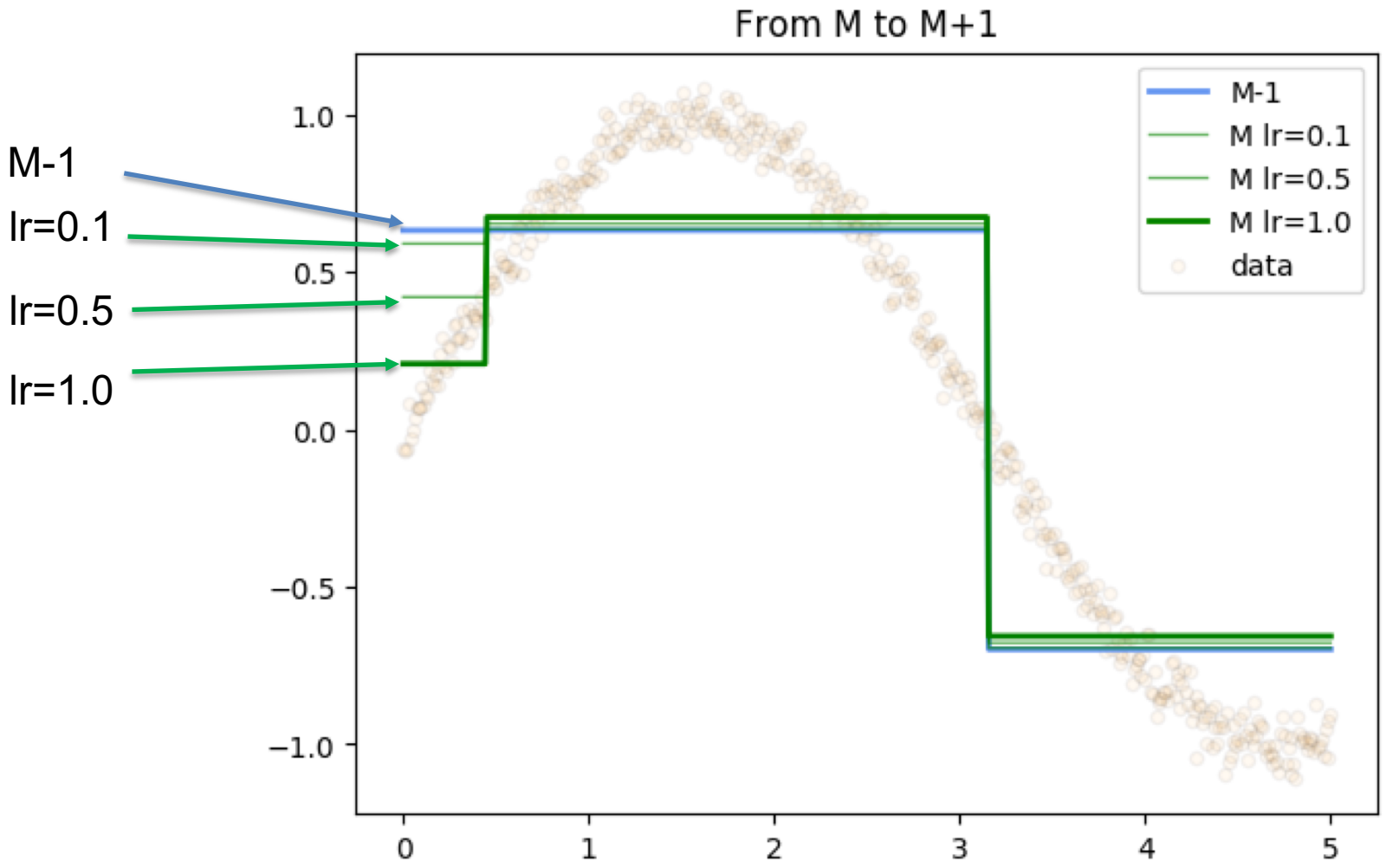
https://github.com/ioskn/mldl_htwg/blob/master/boosting/Least_Square_Boosting.ipynb

Effect of learning rate



Both learning rate and early stopping needs to be tuned for optimal result

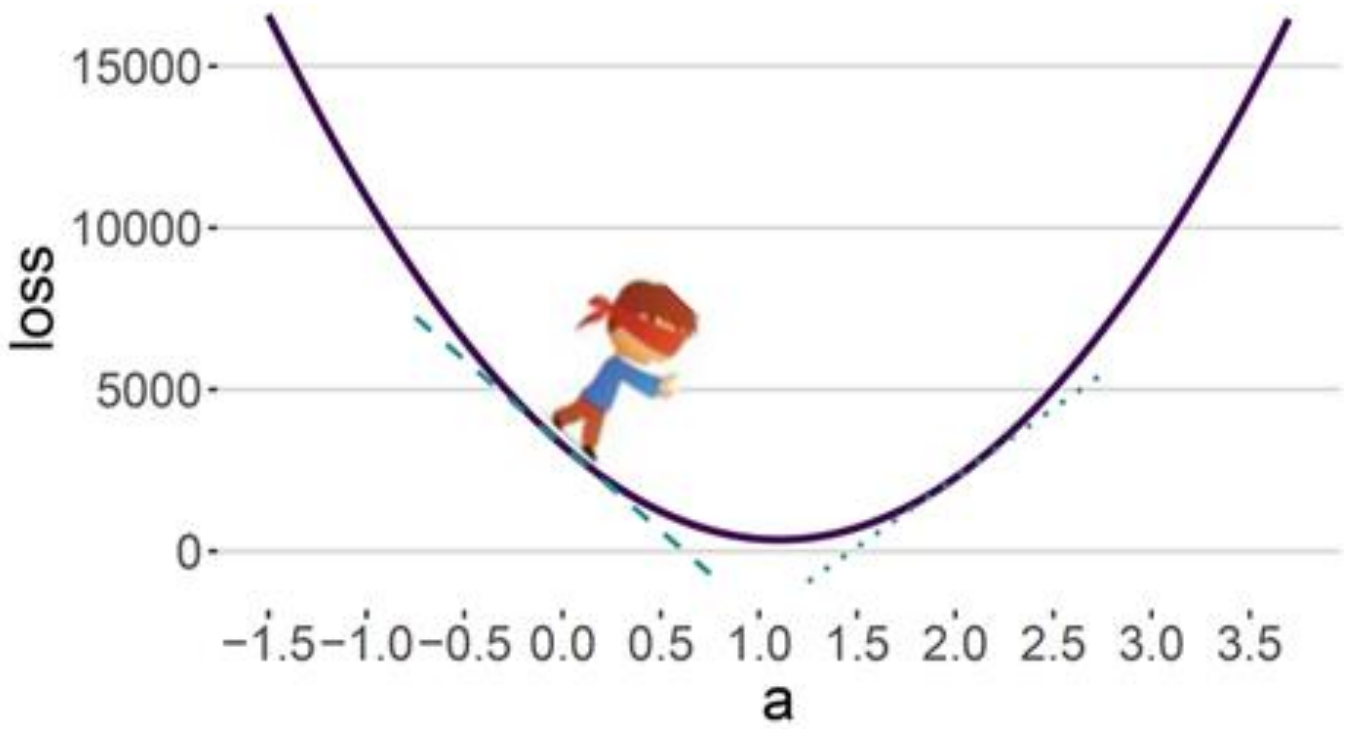
Learning rate as partial step to best solution



We move in the direction of the optimal fit (lr=1.0). But, we do not go the full step.

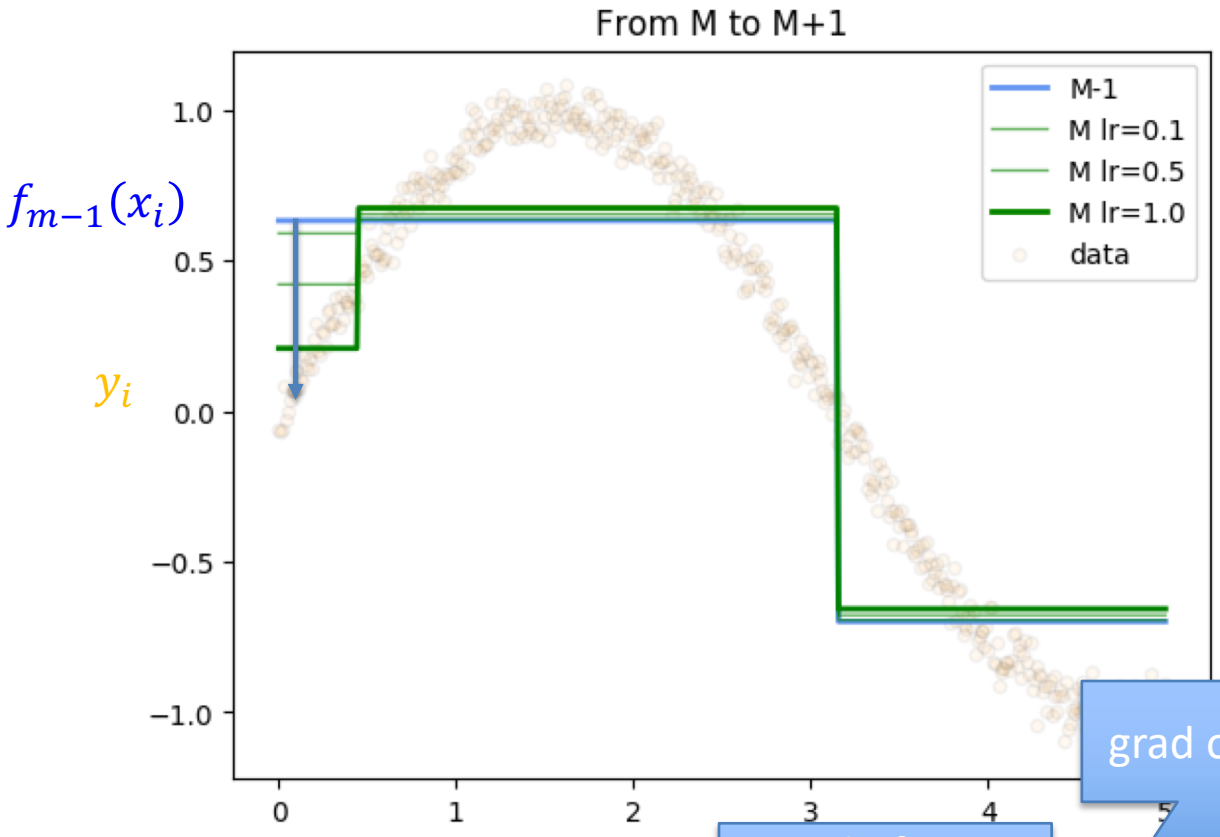
We move in the direction of the optimal fit

- Does this rings a bell?



The gradient in gradient boosting

Connection between gradient and residuals



$f_{m-1}(x_i)$

y_i

grad of MSE(fancy written)

grad of MSE

$$f_{m-1}(x_i) - y_i = w_i - y_i = \frac{\partial}{\partial w_i}\frac{1}{2}(w_i - y_i)^2 = -\frac{\partial}{\partial f_{m-1}(x_i)}\frac{1}{2}L_{MSE}(y_i, f_{m-1}(x_i))$$

w_i (just a variable)

With a squared loss: The residuals are the negative gradients of the loss.

Gradient Boosting

- In gradient boosting residuals are replaced by the gradients of the loss function
- For MSE this lead to the residuals
- For other loss functions generalized residuals are created. These point to the direction in with to move to improve the loss.

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$$

- A tree is fitted to the residuals from the trees of stage *m-1*
  - *This is in contrast to neural networks and most other models in which the gradients are propagated into the model*

TABLE 10.2. *Gradients for commonly used loss functions.*

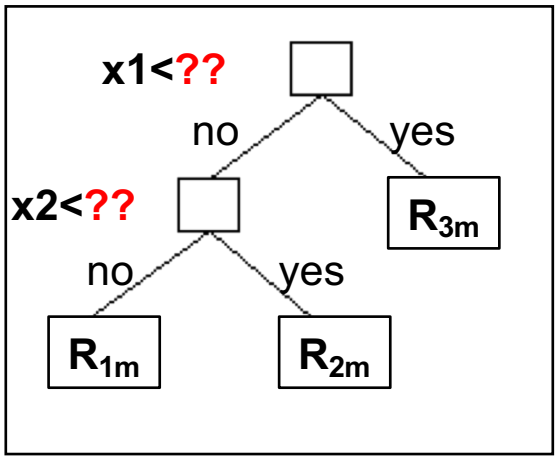
Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$

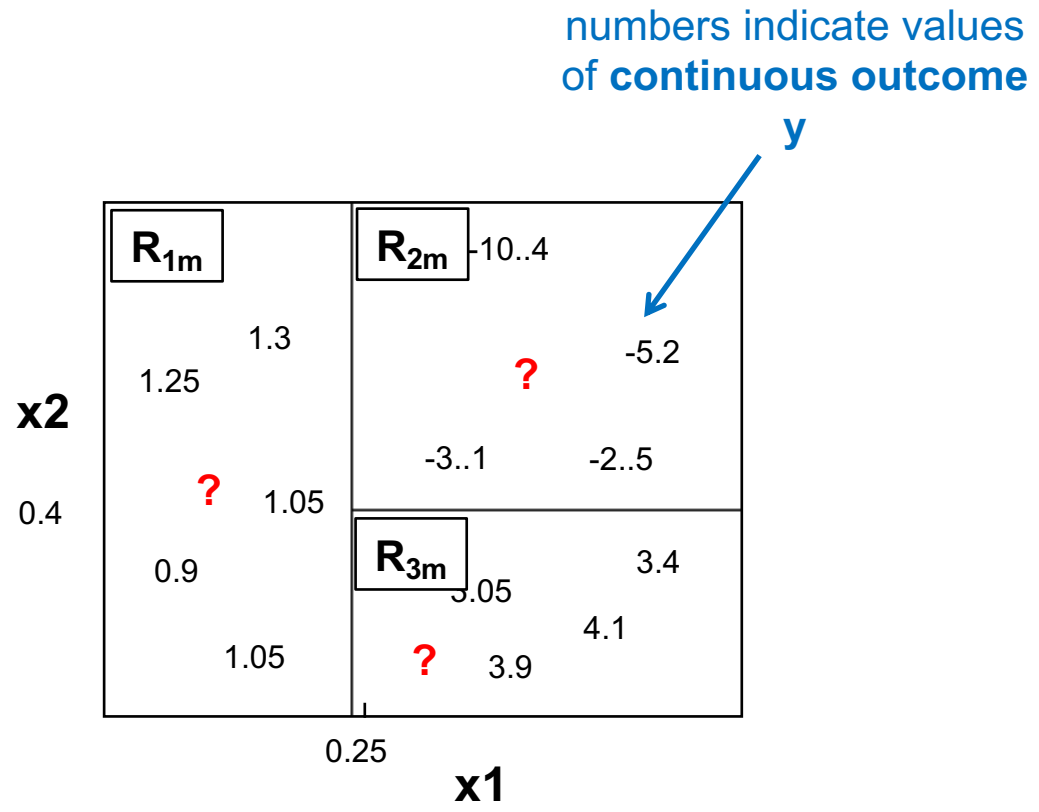
Recap How to find the tree structure of a regression tree?

- Starting with a single region -- i.e., all given data
- At the m-th iteration:

for each region R
  for each attribute x_j in R
    for each possible split s_j of x_j
      record change in score when we partition R into R^l and R^r
Choose (x_j, s_j) giving maximum improvement to fit
Replace R with R^l; add R^r

Score:
- **E.g. based on MSE (mean squared error)**
- **E.g. based on**





What to report in the regions?

In full detail (for regression)

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.

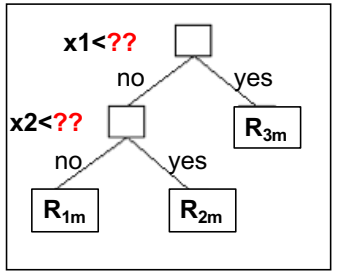
   A single value for all x, e.g. constant

2. For $m = 1$ to M:

   Computation of the generalized residuals to previous stage.

   (a) For $i = 1, 2, \ldots, N$ compute

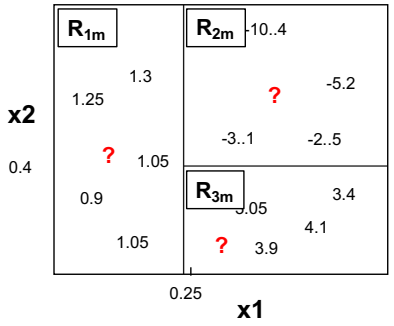
$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$



   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.

   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$



   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.

   The new tree

3. Output $\hat{f}(x) = f_M(x)$.

2b) This tree fitting part involves heuristics and usually does not use the exact loss function. 2c) For MSE this is simply the mean

Loss functions for regression

- MSE
  - Default in most gradient boosting frameworks, NLL of Gaussians
  - loss = $\frac{1}{N}\sum\left(y_i - f(x_i)\right)^2$
- MAE
  - More robust against outliers
  - loss = $\frac{1}{N}\sum|y_i - f(x_i)|$
- Poisson
  - Modeling count data
  - loss = NLL = $-\frac{1}{N}\sum_i \log(p(y_i)) = \sum_i \exp(f(x_i)) - y_i \cdot f(x_i) + \log(y_i!)$

Const factor can be neglected for optimization

- Quantile
  - Modeling the median (or other Quantiles)
  - Complicated

- For more see e.g. documentation of CatBoost:
  - https://catboost.ai/docs/concepts/loss-functions-regression.html

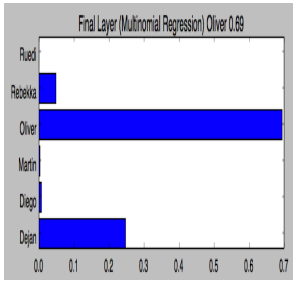
Loss functions for classification with 2 classes

- LogLoss (a.k.a. binary crossentropy)
  - NLL (Default in most gradient boosting frameworks)
  - $p_m(x_i) = \text{sigmoid}(f_m(x_i))$
  - General for one example $p(y_{observed}|x)$
  - Here:
    - $\text{p}(x_i)$ the probability for the 1 class
    - $y_i = \{0,1\}$ is the observed class
  - loss=$\frac{1}{N}\sum_i y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$
  - Same as in logistic regression
- Other measures
  - AUC
  - 1-Acc

For more see e.g.: https://catboost.ai/docs/concepts/loss-functions-classification.html

Loss functions for classification with more than 2 classes

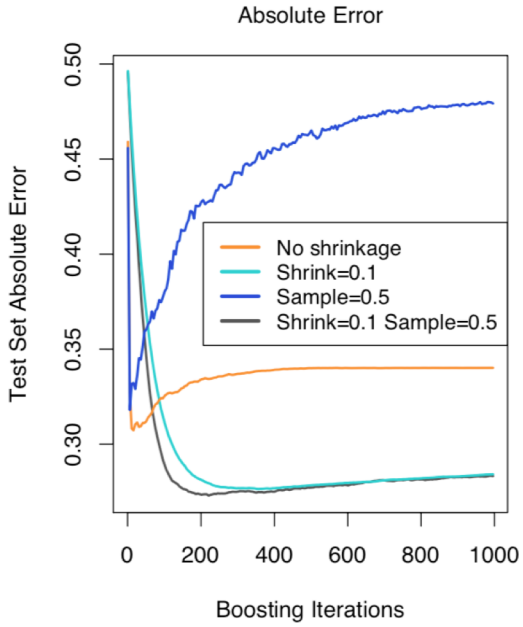
- In classification $f_{mk}(x_i)$ is per class $k = 1 \ldots K$
  - See ESL Algorithm 10.4 for details
- Transformation into a probability
  - $p_{mk}(x_i) = \frac{\exp(f_{mk}(x_i))}{\sum_k \exp(f_{mk}(x_i))}$ is now a probability between 0 and 1
- Loss NLL of the right class (crossentropy)
- Likelihood
  - Generell $p(y_{observed}|x)$
  - Here Loss = $-\sum_{ik} \mathrm{I}(\mathrm{k} = \mathrm{y_i}) \log(p_{mk}(x_i))$
- Example



Example (Face Rec):
Look at class of single training example. Say it's Dejan, if classified correctly p_dejan = 1 ➔ Loss = 0. Real bad classifier put's p_dejan=0 ➔ Loss = Inf.

For more https://catboost.ai/docs/concepts/loss-functions-multiclassification.html

Additional Regularization



ESL 10.12

- So far
  - Early Stopping
    - Control via validation set
  - Learning rate a.k.a. shrinkage
  - gamma [default=0][range: (0,Inf)] minimum loss reduction rec
    Can be increased to fight overfitting with shallow trees.
  - max_depth maximum depth of a tree [default=6][range: (0,Inf)
    Can (often should) be lowered to prevent overfitting.
- Subsampling (rows)
  - In each iteration *m* only a fraction of the data is used
  - Similar to bagging
- Subsampling (columns)
  - Similar to the random forest trick
- Catboost / XGBoost
  - Have additional tuneing parameters
  - See e.g. https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster
- Use crossvalidation for tuning

Boosting Frameworks

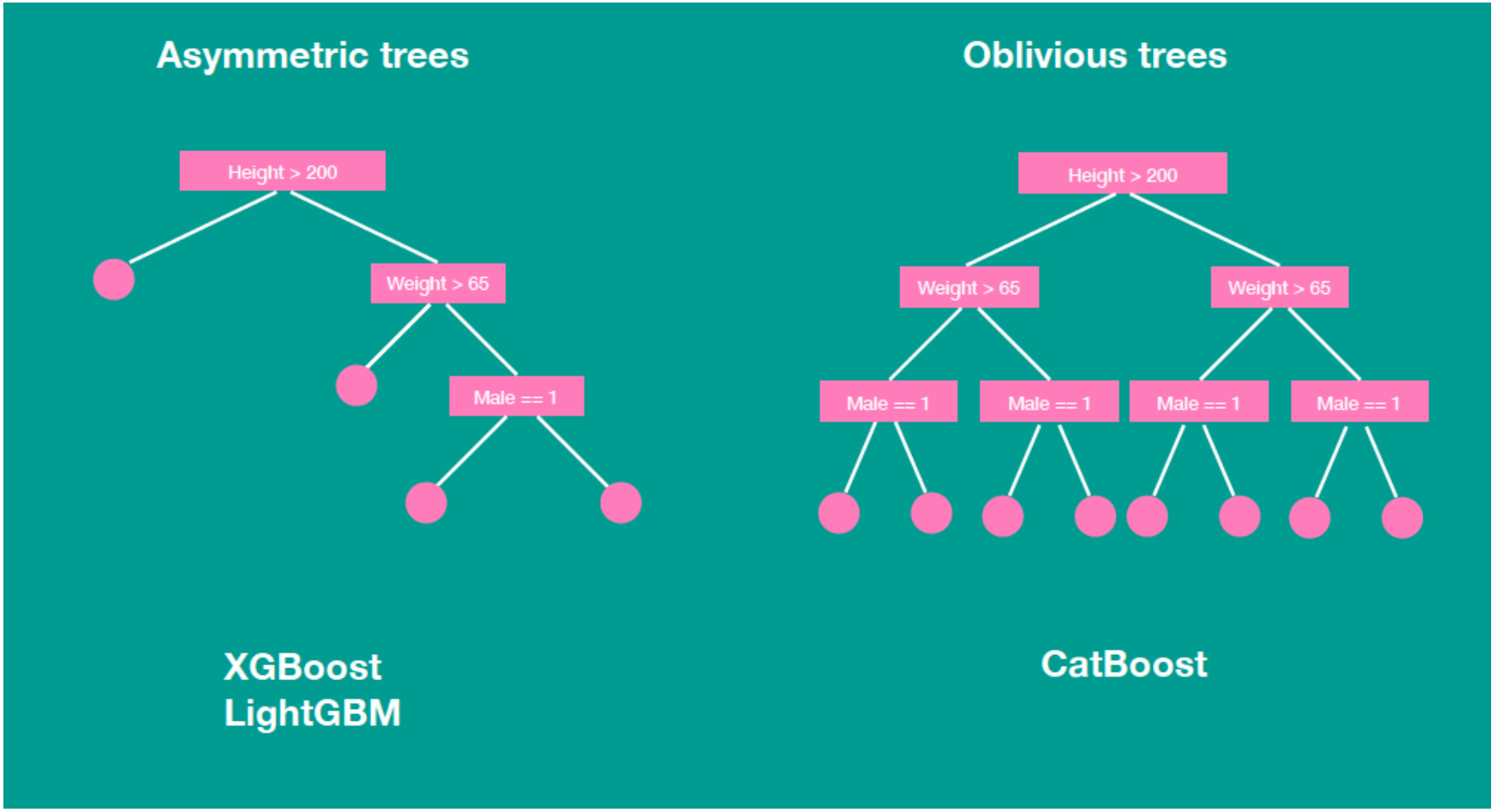
- Extreme Gradient Boosting (xgboost - a particular implementation of GB)
  - Tianqi Chen (2014 code, 2016 published arXiv://1603.02754)
  - Often used in Kaggle Competitions as part of the winning solution

- Other framworks
  - H2O and LightGBM

- CatBoost (Yandex arXiv://1706.09516)
  - Categorical data
  - Ordered
  - Oblivious Trees (next slide)
  - Additional loss functions for ranking

All frameworks work in python and R

Oblivious Tress



- Same splitting criterion is used across an entire level of the tree
- Balanced, less prone to overfitting, and allow speeding in prediction at testing time.

Performance (NLL) for different problems

	CatBoost		LightGBM		XGBoost		H2O	
	Tuned	Default	Tuned	Default	Tuned	Default	Tuned	Default
Adult	**0.26974**	0.27298 +1.21%	0.27602 +2.33%	0.28716 +6.46%	0.27542 +2.11%	0.28009 +3.84%	0.27510 +1.99%	0.27607 +2.35%
Amazon	**0.13772**	0.13811 +0.29%	0.16360 +18.80%	0.16716 +21.38%	0.16327 +18.56%	0.16536 +20.07%	0.16264 +18.10%	0.16950 +23.08%
Click prediction	**0.39090**	0.39112 +0.06%	0.39633 +1.39%	0.39749 +1.69%	0.39624 +1.37%	0.39764 +1.73%	0.39759 +1.72%	0.39785 +1.78%
KDD appetency	0.07151	**0.07138** –0.19%	0.07179 +0.40%	0.07482 +4.63%	0.07176 +0.35%	0.07466 +4.41%	0.07246 +1.33%	0.07355 +2.86%
KDD churn	**0.23129**	0.23193 +0.28%	0.23205 +0.33%	0.23565 +1.89%	0.23312 +0.80%	0.23369 +1.04%	0.23275 +0.64%	0.23287 +0.69%
KDD internet	**0.20875**	0.22021 +5.49%	0.22315 +6.90%	0.23627 +13.19%	0.22532 +7.94%	0.23468 +12.43%	0.22209 +6.40%	0.24023 +15.09%
KDD upselling	**0.16613**	0.16674 +0.37%	0.16682 +0.42%	0.17107 +2.98%	0.16632 +0.12%	0.16873 +1.57%	0.16824 +1.28%	0.16981 +2.22%
KDD 98	**0.19467**	0.19479 +0.07%	0.19576 +0.56%	0.19837 +1.91%	0.19568 +0.52%	0.19795 +1.69%	0.19539 +0.37%	0.19607 +0.72%
Kick prediction	**0.28479**	0.28491 +0.05%	0.29566 +3.82%	0.29877 +4.91%	0.29465 +3.47%	0.29816 +4.70%	0.29481 +3.52%	0.29635 +4.06%

https://towardsdatascience.com/introduction-to-gradient-boosting-on-decision-trees-with-catboost-d511a9ccbd14

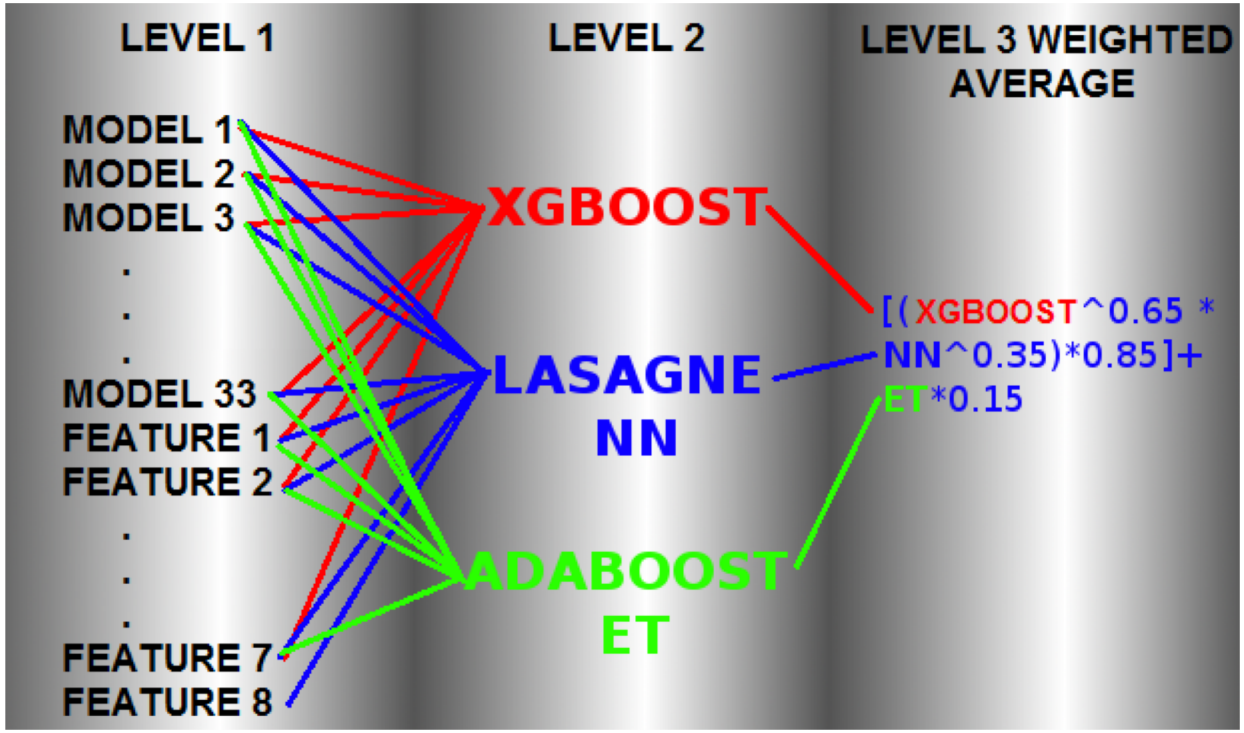
Demo time

- https://github.com/ioskn/mldl_htwg/blob/master/boosting/CatBoost.ipynb

How to construct the best performing model?

Join Forces, use different models including ensemble methods and learn how to combine their predictions for a joint prediction.

Winning solution for the Otto Challenge



Use outcome from models (e.g. xgboost) as meta features

See: https://www.kaggle.com/c/otto-group-product-classification-challenge/forums/t/14335/1st-place-winner-solution-gilberto-titericz-stanislav-semenov/79598#post79598